Breakout

100004004

April 20, 2011

Contents

1 Introduction **2**
- 1.1 A little background . . . 2
- 1.2 Task . . . 2

2 Specification **3**

3 Design **4**
- 3.1 Structure . . . 4
  - 3.1.1 View . . . 4
  - 3.1.2 Model . . . 4
  - 3.1.3 Controller . . . 4
  - 3.1.4 Everything else . . . 5
- 3.2 Ball Movement . . . 5
- 3.3 Collision Detection . . . 5
- 3.4 Level Loading . . . 5

4 Usage **6**
- 4.1 Screenshot . . . 6
- 4.2 To Run . . . 7

5 Lessons learned **8**

Chapter 1

Introduction

1.1 A little background

The game "Breakout" originated from the ancient age of arcade games, a time in which people would aimlessly fiddle with a mechanical stick in order to produce different series of lights from a box within a cabinet. It was commissioned by Atari, as a single player alternative to the popular multiplayer game "Pong", to a certain man who would later be known to be one of the most influential men in the computing and tech industry; and of which he was first recognised to have demonstrated his excellent power at exploiting other people.

Breakout consists of a wall of bricks, of which a ball is hit against. By hitting the ball against the wall, the bricks would gradually be eroded, allowing for the ball to "breakout". The player manages to rack up points by destroying as many bricks as possible, all the while preventing the ball to hit the floor, by using their bat.

1.2 Task

The aim of this was to write a clone of the game in Java, and possibly improve upon it. A working game sshould be produced in the style of Breakout or later, more advanced clones such as Arkanoid which introduced exciting new extras such as powerups and advanced level design.

Chapter 2

Specification

The specification attempted includes the following:

1. Bricks, Paddle and ball making up the well known basics of the game
2. A scoring mechanism of one's own devising
3. Change in paddle size
4. Changes in ball speed
5. Changes in ball angle, depending on where it hits the paddle
6. Extra levels with different arrangements of bricks
7. "Powerups" modifying behaviour

Some slight changes from the spec:

1. Model View Controller pattern instead of Model Delegate
2. "Powerups" and paddle size changes are done on completion of levels
3. ball speed, whilst changeable, happens as part of a vector design as opposed to a separate variable

Chapter 3

Design

3.1 Structure

The structure is a Model View Controller design. This allows for a logical split of various parts of the program, and was well suited for the Light Weight Java Game Library, an OpenGL and OpenAL wrapper for Java that was chosen for this project.

3.1.1 View

The view is where everything rendered from. This was done using OpenGL, with rectangles represented using vertices (and rendered technically with triangle fans, as graphics hardware tends to use triangles rather than quadrilaterals), stored in Vertex Arrays. This allowed for re-use, and instead of specifying exactly where to draw them on the screen, it allowed for relative dimensions, then to be translated later using a translation matrix.

3.1.2 Model

The model is where all current state data was kept everything from the score to exact position of each element on the screen. It holds references to every game object, and initialises calls to the controller. Different levels are loaded from files here, with the model populating the ArrayList representing all the blocks.

3.1.3 Controller

The controller is where all the updates to the model's data take place, taking care of calculations of where the ball should be next, and taking input and moving the paddle's co-ordinates accordingly.

3.1.4 Everything else

The Main class is a small class that created the initial Model, View and Controller objects. The Game Objects were all extended from a "CollidableObject" class. This allowed for the collision detection to be abstracted away from the workings, and allowed for a relatively clean structure, allowing for individual features on certain class.

3.2 Ball Movement

The ball was designed to move as a vector. This was done for flexibility direction could easily be changed, and the x and y components were separate allowing for easing reflection, amongst other things. This also meant that speed was variable, depending on how far the ball was moving in one axis. The direction of which a ball was reflected off a paddle was a function of how far along the paddle it was from the centre as such, this means that larger paddles means faster balls.

Issues were raised by this use of movement as it is moving every update, as opposed to a fixed length of time, it can sometimes speed up and down. One way of solving this is to multiply the change by a delta time since the last update. However, doing ths caused major issues if no time had passed (in milliseconds), then it would multiply by 0, meaning that it could stay in the same place whilst colliding, and thus collide multiple times, ruining the accuracy of the score and increasing the likelihood of the ball reacting in unrealistic ways (such as moving at $4x10^9$ pixels a second). Eventually such a method was simply scrapped.

3.3 Collision Detection

Simple bounding box collision detection was used. This appeared to be both fast and adequate for the job, and as such seems like a good a choice.

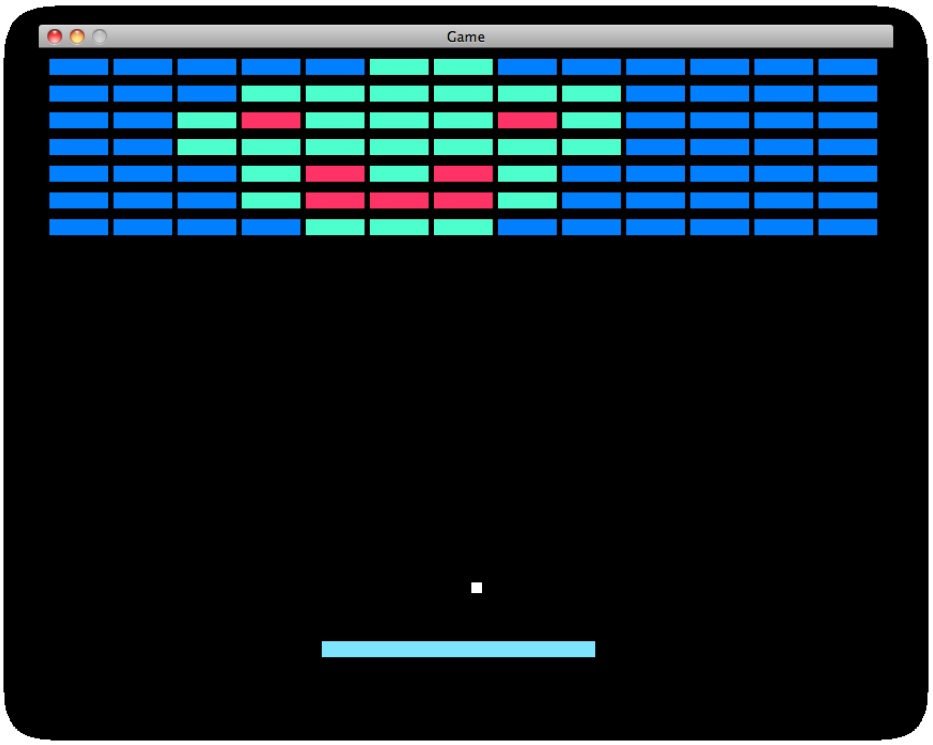
3.4 Level Loading

Level loading was done by specifying a grid of 13 columns of identical blocks, differing only in colour and in difficulty to destroy. This was loaded using GSON, a JSON reading library created by Google (advantages include support for generic types, useful for ArrayLists¡Generic Type¿.

Chapter 4

Usage

4.1 Screenshot



4.2 To Run

It should be fairly simple to run the game, as I have pieced together script files for various platforms. From a command line, either run ./main for linux mac platforms, or run main.bat on a windows platform.

Left click to start the ball going, right click to reset. Press left click during game to cheat for testing purposes.

Chapter 5

Lessons learned

The most valuable techniques I learned in this particular project was the use of the OpenGL graphics library (if you allow the tautology). Not a simple library to learn, and one in which nearly all documentation provided online is out of date, or not usable within Java. This also encompassed learning various other things, such as Java's new I/O library use with channels and buffers (java.nio). A missed update call on one revision of code crashed the computer and proceeded to flood the screen with random pixels from the screenbuffer, which was a difficult bug to diagnose (switching between two computers whilst changing one line at a time discovered it in a couple of days).

Next time I have an opportunity to use OpenGL, I would like to build upon my knowledge by including textures on pictures, and learning the shader language. This would allow for a much prettier and presentable game, and could have allowed for special effects when things such as powerups were collected. Also, I couldn't fix issues with on screen text satisfactorily enough to leave in the game and as such things such as the score are printed on the command line instead, which leads to a disjointed experience.