SIENA*college*
Computer Science

Computer Science 225
Advanced Programming
Siena College
Spring 2020

# Topic Notes: Animation with Collisions

These notes point out some features of an example that includes *collision detection* between graphical objects.

`https://github.com/SienaCSISAdvancedProgramming/Breakout`

This is a simple "Breakout" game, where one or more balls bounces around a room, hitting, bouncing off of, and removing bricks, with a paddle to keep the balls in the playing area. This program requires collision detection between the ball and the paddle, and the ball and the bricks.

The only collision detection implemented is to see if a circle (the `BreakoutBall`) and a rectangle overlap in the Euclidean plane, as this is what is required to detect overlap between the ball and either the paddle or a brick. This is accomplished with a `static` method in the `Collision` class, which implements the algorithm as described at `https://www.gamedevelopment.blog/collision-detection-circles-rectangles-and-polygons/`. We can easily enough write similar methods to detect overlap for other shapes.

Try out the `CollisionTester` program in this repository. It draws a square and a draggable circle. Each time the window is painted, the circle is green if it is not overlapping the square, red if it is. Try it out - this algorithm is nice and precise.

---

## A Game Using Collision Detection

The `Breakout` program is playable, but its video game physics leaves something to be desired, and lacks many features that would not be too difficult to add that would make the game much more fun. But it serves our purposes.

The program has a number of other items of interest, including:

- It uses the `threadgraphics` package to simplify a bit of the code in the `Breakout` and `BreakoutBall` classes. Note that we use the `buildGUI` method also to construct the `BreakoutPaddle` and `BrickCollection` classes, and to create a repaint thread so we don't need to call `repaint` in various places throughout the code.
- Separate classes are used to manage the panel and creation of new `BreakoutBalls` (`Breakout`), to handle the life of one ball (`BreakoutBall`), to manage the paddle (`BreakoutPaddle`), and to manage the bricks (`BrickCollection`).
- The only mouse event handler in `Breakout` is `mousePressed`, which unconditionally creates a new `BreakoutBall`. Nice improvements might be to allow only one (or a small number) of balls to be in play concurrently, or to limit to a total number of balls before the game ends.

- The `BreakoutPaddle` repositions and paints the paddle, making sure it follows the x-coordinate of the mouse pointer, but not letting the paddle leave the bounds of the window. Note that this program has different mouse events delivered to different classes. We could have put the `mouseMoved` into `Breakout`, but then it would have had to call a method of `BreakoutPaddle` to reposition it. As it is, we have the `mouseMoved` right in `BreakoutPaddle` where we need it.

- The `BrickCollection` class is responsible for drawing the bricks, for knowing which ones have been hit and destroyed (hence should no longer be drawn), and for checking which brick, if any, is overlapping with the ball.

- In many ways, it's the `BreakoutBall` class that drives the action of the game. As each ball can move on its own, it needs to be able to check not only if it has hit the walls like we saw in the `BallTosser`, but if it has hit the paddle or any of the bricks, and needs to respond appropriately in each case. Therefore, it needs to know about the `BreakoutPaddle` and `BrickCollection`, a reference to each of which is passed to `BreakoutBall`'s constructor and stored in instance variables. Every time the `BreakoutBall` moves, it calls `BreakoutPaddle`'s `overlapsBall` method to see if it should bounce off the paddle, and `BrickCollection`'s `hitBrick` method to see if it has hit and bounced off a brick.