# Collisions

I have noticed a couple of questions on the community forum about collision detection and thought I would write a simple program to demonstrate the concepts.

The code for the application is shown below. It starts by drawing a number of fixed items on the screen. Pressing the ‘A’ button causes two rectangular images to cross the screen colliding with both the fixed objects and each other. The status of the collisions is logged out to the serial port so you can see the results.

#include <Arduboy2.h>

#include "Images.h"

Arduboy2 arduboy;

Sprites sprites;

int16\_t xA = -16, yA = 12, xB = 128, yB = 24;

int16\_t frame;

void setup() {

arduboy.begin();

}

void loop() {

arduboy.clear();

Serial.println("------------------------");

Serial.print("A collides with circle : ");

Serial.println(arduboy.collide( (Rect){xA, yA, 16, 16}, (Rect){24, 4, 16, 16} ));

Serial.print("A collides with B : ");

Serial.println(arduboy.collide( (Rect){xA, yA, 16, 16}, (Rect){xB, yB, 16, 16} ));

Serial.print("A collides with dot : ");

Serial.println(arduboy.collide( (Point){64, 20}, (Rect){xA, yA, 16, 16} ));

Serial.print("B collides with square : ");

Serial.println(arduboy.collide( (Rect){xB, yB, 16, 16}, (Rect){96, 32, 16, 16} ));

Serial.print("B collides with dot : ");

Serial.println(arduboy.collide( (Point){64, 32}, (Rect){xB, yB, 16, 16} ));

arduboy.drawCircle(32, 12, 8, WHITE);

arduboy.drawRect(96, 32, 16, 16, WHITE);

arduboy.drawPixel(64, 20, WHITE);

arduboy.drawPixel(64, 32, WHITE);

sprites.drawOverwrite(xA, yA, squareA, frame);

sprites.drawOverwrite(xB, yB, squareB, frame);

xA++; if (xA >= 128) { xA = 0; }

xB--; if (xB < -16) { xB = 128; }

arduboy.display();

arduboy.delayShort(10);

while (true) {

if (arduboy.pressed(A\_BUTTON)) { break; }

arduboy.delayShort(20);

}

}

But how does the collision detection work?

The Arduboy2 library provides two collision detection methods which are shown below:

bool collide(Point point, Rect rect);

bool collide(Rect rect1, Rect rect2);

It also provides two structures to assist:

struct Rect {

int16\_t x;

int16\_t y;

uint8\_t width;

uint8\_t height;

};

struct Point {

int16\_t x;

int16\_t y;

};

The first function tests to see if a point is contained within the specified rectangle. The sample code below demonstrates this:

Point p;

Rect r;

p.x = 10;

p.y = 10;

r.x = 5;

r.y = 5;

r.width = 10;

r.height = 10;

Serial.println(arduboy.collision(p, r)); < Will print 1 as the point at position 10, 10 is in the centre of the square whose top left corner is at position 5, 5 and lower right corner is at 15, 15.

p.x = 20;

p.y = 20;

Serial.println(arduboy.collision(p, r)); < Will print 0 as the point at position 20, 20 is outside of the square whose top left corner is at position 5, 5 and lower right corner is at 15, 15.

The second function compares the position of two rectangles and tests to see if there is any overlap. This is demonstrated in the code below:

Rect r1 = {12, 8, 16, 16};

Rect r2 = {16, 20, 16, 16};

Serial.println(arduboy.collision(r1, r2)); < Will print 1 as the two rectangles overlap. The first rectangle’s top left corner is at 12, 8 and its lower right corner is at 28, 24. The second rectangle’s top left corner is at 16, 20 and its lower left corner is at 32, 36.

r1.x = 40;

r1.y = 40;

Serial.println(arduboy.collision(r1, r2)) < Will print 0 as the two rectangles do not overlap anymore. The first rectangle’s top left corner is now at 40, 40 and its lower right corner is at 56, 56. The second rectangle’s top left corner is at 16, 20 and its lower left corner is at 32, 36.

One (minor) issue with the detection code is that when detecting collisions, you need to construct the ‘rectangle’ and ‘point’ structures based on the positions of the players and obstacles in the game.

Serial.print("A collides with B : ");

Serial.println(arduboy.collide( (Rect){xA, yA, 16, 16}, (Rect){xB, yB, 16, 16} ));

Take the example code shown above which was taken from my sample application. The code anonymously populates two rectangle structures with the current positions of the moving squares and the image size (16 x 16 pixels). This is relatively simple when there are only a few objects to consider but can become a little more complex and error prone if there are numerous objects with different sizes.

A typical design pattern for action games is to define a class to hold information about players including their current position. A sample header and implementation class is shown below

Player.h:

#include "Arduboy2.h"

class Player {

public:

Rect getRect();

int16\_t x;

int16\_t y;

const uint8\_t \*bitmap;

};

Player.cpp:

#include "Player.h"

#include "Arduboy2.h"

Rect Player::getRect() {

return (Rect){x, y, pgm\_read\_byte(bitmap), pgm\_read\_byte(bitmap + 1)};

}

The header declares three public variables for the player’s coordinates and the image used when rendering the screen. It also prototypes a function that will return a rectangle structure that will describe the area on screen that the player currently occupies. The implementation of that function is included in the .cpp file. It simply creates and returns a populated *Rect* structure using the ‘x’ and ‘y’ coordinates from the class itself and the width and height of the image by reading these from the image array itself.

The code below shows the declaration of two player objects and shows how they can be used when testing collisions.

Player playerA;

Player playerB;

playerA.x = 20;

playerA.y = 12;

playerA.bitmap = squareA;

playerB.x = 100;

playerB.y = 24;

playerB.bitmap = squareB;

...

Serial.print("A collides with B : ");

Serial.println(arduboy.collide(playerA.getRect(), playerB.getRect()));

In my opinion, this approach results in much more readable code.

The simple collision application and the ‘advanced’ version using a ‘Player’ class can be found in my repository.

<https://github.com/filmote/Collide_Article1> (simple example)

<https://github.com/filmote/Collide_Article2> (using the ‘Player’ class).

Happy coding!