Tap Tap Tap

It's time to take a break from our Ball game so we can learn about sound and images.

Before we can do that, we need a new type of gesture needed by a drum machine
Tap – a single event which indicates where the finger tapped on the screen

We will add the new gesture code in main() just like we did for drag
It will have a callback method in our game similar to the ones for drag
Any component can check to see if it has been tapped on

We will make components that are images
- Different percussive instruments that make their sound when tapped
- A record button to record a drum track
- A play button to stop playing the recorded track
- A stop button to stop playing or recording

Add this code to the main() method in main.dart

Before runApp

```
TapGestureRecognizer tapper = TapGestureRecognizer();
tapper.onTapDown = game.onTapDown;
```

After runApp

```
flameUtil.addGestureRecognizer(tapper);
```

And in the game, we add the onTapDown() method to record the tap

```
bool wasTapped = false;  // set to true when screen tapped
double tapX;   // x coordinate uf oser's finger
double tapY;   // x coordinate uf oser's finger


void onTapDown(TapDownDetails d) {
  tapX = d.globalPosition.dx;
  tapY = d.globalPosition.dy;
  wasTapped = true;
}
```

Any component can check the game instance variables to see where the screen was tapped.

A component will do the following to check that it has been tapped

- Check every time update() is called
- Check the game wasTapped variable. If it is false, no tap happened
- If it is true, check the tapX and tapY coordinates to see if they are inside the component
- If the tap was inside the component, set wasTapped back to false so that no more checking will be done by other components
- If the tap was outside the component, leave the variable on so other components can check
- If the component is the one tapped, do whatever is needed to handle the tap like playing a sound

Any component that fits inside a rectange on the screen is a good candidate for taps
Images have a well-defined area on the screen where the tap can be inside or outside

Images

We will make a new component class called Drum. It is made from a new type of component class called SpriteComponent. A sprite is an image that can be moved around the screen. Make a new file drum.dart in the components folder

```
import 'dart:ui';
import 'package:flame/components/component.dart';
import 'package:flame/flame.dart';
import 'package:flutter_ball/flutterball_game.dart';

class Drum extends SpriteComponent  {
  Rect hitRect;
  final FlutterballGame game;
  final String sound;

  Drum(this.game, double x, double y, double size, String image, this.sound) :
super.square(size, image) {
    this.x = x;
    this.y = y;
    hitRect = Rect.fromLTWH(x, y, width, height);
    Flame.audio.play('bounce.wav');  // play once so it's ready for taps
  }
}
```

We have our instance variables

- game is the FlutterballGame object like we use in other components
- hitRect is the rectangle that encloses the image. If a tap “hits” inside this rectangle, we play the drum sound
- sound is the name of the mp3 or wav file we want to play when the drum is tapped

The information needed to set these variables is passed into the constructor.

```
Drum(this.game, double x, double y, double size, String image, this.sound) :
super.square(size, image) {
  this.x = x;
  this.y = y;
  hitRect = Rect.fromLTWH(x, y, width, height);
}
```

Notice we set an x and y variable that we haven't defined. They are actually defined in the SprintComponent class that this class is based off of. This is where the game engine will draw the image.

We create the sprite image by passing the size and image into the square constructor of the SpriteComponent. The width and height variables are also defined and set by the SprintComponent once the image is passed into the constructor.

Our update() method is where we check to see if the image was tapped.

```
void update(double t) {
  if (game.wasTapped && hitRect.contains(Offset(game.tapX,game.tapY))) {
    game.wasTapped = false;  // reset tap
    Flame.audio.play('bounce.wav');
  }
}
```

We check if a tap was done and if the tap is inside our hit rectangle. The Rect class has a handy method called contains() that tells us if a point is inside or not. We need to turn the tap location x and y into a single point (offset) that we pass to the contains() method.

The Flame game engine has a handy way of playing audio by passing the file name to audio.play.

We need to get those image files and audio files into our Flutter program.

Assets

An asset is any file that is needed by our program. These files can be images, sound, or other data files. In our case, we have a number of images and sounds we need to package with our game. We store these assets in a folder in our project called assets.

You can go to GitHub and copy down the assets folder from there to your project. It has all of the images and sounds for the drum machine.

https://github.com/shawnlg/flutter_ball/tree/07_one_drum

Most of our sounds are in assets/audio/drum. We have one more sound, bounce.wav that will be used in our game when the ball bounces.

The images are all in assets/images/drum

Once you put the images and sounds in your project, you need to tell Flutter about them.

We tell Flutter about our assets by listing them in the pubspec.yaml file. They are listed under the flutter: line like this

```
flutter:
  uses-material-design: true
  assets:
    - assets/audio/bounce.wav
    - assets/images/drum/bass.jpg
    - assets/audio/drum/bass.wav
    - assets/images/drum/bongos.png
    - assets/audio/drum/bongos.wav
   …
```

You can add in all of the images and sounds manually or just copy them from the file in GitHub.

Finally, to make the images and sounds load as fast as possible, you tell Flutter to load them all into memory by listing them in the main() method.

The list of assets is fairly long which makes the main() method long. Copy them from the existing main() method in GitHub.

```
FlutterballGame game = FlutterballGame();

// load assets
Flame.images.loadAll(<String>[
  'drum/cowbell.jpg',
  'drum/cymbols.jpg',
  'drum/echo.jpg',
  'drum/frame drum.jpg',
  'drum/maracas.jpg',
  'drum/open hat.jpg',
  'drum/tambourine.jpg',
  'drum/triangle.jpg',
  'drum/whistle.jpg',
  'drum/woodblock.jpg',
]);

Flame.audio.disableLog();
Flame.audio.loadAll(<String>[
  'bounce.wav',
  'drum/cowbell.wav',
  'drum/cymbols.wav',
  'drum/echo.wav',
  'drum/frame drum.wav',
  'drum/maracas.wav',
  'drum/metro main.wav',
  'drum/metro off.wav',
  'drum/open hat.wav',
  'drum/tambourine.wav',
  'drum/triangle.wav',
  'drum/whistle.wav',
  'drum/woodblock.wav',
]);
```

Testing our Drum

We will add a single drum to our game to test out that our component is working. We do it the same way we have tested our other components, we manually add it to our game in flutterball_game.dart

Inside the constructor we have

```
// make a new drum game component
var drum = Drum(this,50,50,100,'drum/frame drum.jpg','drum/frame drum.wav');

// tell the game about this component
add(drum);
```

We manually place a single image at location 50,50 on our screen.

We pass the location, width, image file, and audio file to our Drum() class and it creates the component. We add it to the game.

Playing the Drum

Run the game on your phone or emulator. Remember, if there are problems, you can find the working code at

https://github.com/shawnlg/flutter_ball/tree/07_one_drum

If all goes well, tapping on the drum will play the sound. Tapping anywhere else will do nothing.

Note – if using an emulator, you might get delayed sound when tapping on the drum. You will get much better results on a real phone.

If you'd like, put some other drum components on the screen with different images and sounds. Test that they can all respond to the taps correctly.

Next, we will make an actual drum machine component that adds all the drums, records, and plays drum patterns.