A Better Drum Machine

We have a drum machine of sorts
Tap a drum and it makes a sound
We can do better than that

We could improve things by adding all the drum components to the game. That would actually be very easy – just add more drums in the game constructor.

We want to have more control than just tapping on images and playing their sound. So we'll make a new component called DrumController in a component file called drum_controller.dart

See how much you can do on your own. Create a basic component that has all of the needed methods and a constructor. Give it the normal size variables and game variable like our other components. We'll go over it next.

Oh, and see if you can find the bug in drum.dart. Every drum will play the same sound.

Here's the start of our DrumController component

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

class DrumController extends Component {
  // instance variables
  FlutterballGame game;
  double width=0;  // size of the screen in the x direction
  double height=0;  // size of the screen in the y direction

  // constructor
  DrumController(this.game) : super() {
  }

  void render(Canvas c) => null;

  void update(double t) {
  }

  void resize(Size size) {
    if (size.width == 0) return;  // don't bother

    // save screen width and height
    width = size.width;
    height = size.height;
  }

}
```

There's nothing new in the code so far. Since the drum controller will be using a Drum component, you should add it to the imports.

```
import 'package:flutter_ball/components/drum.dart';
```

We want our game to call the drum controller component instead of just a drum, so that needs changing too.

```
import 'package:flutter_ball/components/drum_controller.dart';
// make a new drum machine game component

var drumMachine = DrumController(this);

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

Notice we called our DrumController drumMachine. That's ok. The object name does not have to be the same as the class name.

Did you find the bug in drum.dart?

```
Flame.audio.play(sound);
```

Unum Variables

We will use a new type of variable called an enumeration (or enum for short)

It's like defining a whole bunch of constants automatically

Study this enum we will be using in the drum controller.

```
enum State {
  STARTUP,   // game is starting up, no size yet
  READY,     // game is ready to start adding components
  TAP,       // in tap mode, just play what is tapped
  RECORD,    // record a drum loop
  PLAY,      // play a recorded drum loop
}
```

The enum goes between the import statements and the class.

Add this instance variable that holds one of the values from the enum

```
State state = State.STARTUP;
```

Can you guess what the state variable will be used for?

State

We have a variable called state that will hold the “state” of the drum controller.

Examples of state:

Sleep state

- Sleeping
- Awake
- Coma

Bank account state

- Closed
- Overdrawn
- Inactive
- Normal

Drum Controller States

Why use states?

- Certain things should only be done once. Like adding drums. Easier than using a bunch of flags
- State values are clear, makes code easy to understand
- Very easy to have different pieces of code for each state

Here's the flow that will change the states

- Default state value is STARTUP – it stays this way until resize() is called. We don't do anything when in this state
- Resize will put us in the READY state. Time to add drums and set things up.
- Once things are set up, we go into the TAP state. In this state, we do nothing, but the drum components will play sounds when they are tapped.
- Tapping the record button will put us in the RECORD state where we keep track of every drum tap
- Tapping the stop button puts us back in the TAP state
- Tapping the play button puts us in the PLAY state where we play back the drum pattern recorded over and over until the stop button is tapped.

Adding Drums

How drums are laid out

- We will thing in percent for placing the drums on the screen

  0% from the top is right at the top. 100% is right at the bottom
- 3 rows of 3 drums each – 9 drums
- First row will start 10% of the screen down from the top
- Each row takes up 20% of the height of the screen

  row 2 starts at 30% from the top, row 3 at 50%

  This leaves enough room at the bottom for the buttons
- Equal space between each drum and the left/right edges of the screen
- The size of the drum image is 20% of the width of the screen

Try drawing this out on a piece of paper. Figure out how big the gaps will be between the drums and where each one will be placed in terms of % from the top and % from the left.

To make it easy to lay out our drums, we will define some instance variables that can be used as percentages of the width and height. Add these

```
double pctX=0;   // pixels as a percentage
double pctY=0;   // pixels as a percentage
```

They will be updated in our resize method. That method will also change our state from STARTUP to READY.

```
void resize(Size size) {
  if (size.width == 0) return;   // don't bother

  // save screen width and height
  width = size.width;
  height = size.height;
  pctX = width/100;
  pctY = height/100;
  state = State.READY;
}
```

Finally, so we can keep track of our drums, we need a list of drum objects as another instance variable.

```
List<Drum> drums;
```

Lists

We will eventually need to record our drum taps, so the controller needs an easy way to group all of the drums together and access them. We use a list for this. Here is how we add the 9 drum objects to the list. Add this to the update() method.

```
// what we do depends on the state of the drum machine
switch (state) {
  case State.READY:
    // create the drums
    drums = [
      Drum(game, pctX*10, pctY*10, pctX*20, 'drum/frame drum.jpg', 'drum/frame drum.wav'),
      Drum(game, pctX*40, pctY*10, pctX*20, 'drum/cymbols.jpg', 'drum/cymbols.wav'),
      Drum(game, pctX*70, pctY*10, pctX*20, 'drum/echo.jpg', 'drum/echo.wav'),
      Drum(game, pctX*10, pctY*30, pctX*20, 'drum/woodblock.jpg', 'drum/woodblock.wav'),
      Drum(game, pctX*40, pctY*30, pctX*20, 'drum/maracas.jpg', 'drum/maracas.wav'),
      Drum(game, pctX*70, pctY*30, pctX*20, 'drum/open hat.jpg', 'drum/open hat.wav'),
      Drum(game, pctX*10, pctY*50, pctX*20, 'drum/tambourine.jpg', 'drum/tambourine.wav'),
      Drum(game, pctX*40, pctY*50, pctX*20, 'drum/triangle.jpg', 'drum/triangle.wav'),
      Drum(game, pctX*70, pctY*50, pctX*20, 'drum/whistle.jpg', 'drum/whistle.wav'),
    ];
  default:
}
```

There are a number of things going on here

- We use a switch statement which lets us write different pieces of code for each value of a variable
- We need this because we only want to add the drums when we are ready, and only once.
- Remember – update can be called 60 times a second
- Inside the list, we just do all of our constructors, creating 9 drums
- Notice the percent values for the drum x and y coordinates and the image size
- Each drum object will have a different sound file and image file

Guess what we still need after we add the drums to the list?

- We need to change the state variable, or on the next update we will add the drums again and again
- We need to add all the drums in the list to the game itself

Add this code to change the state after you make the drum list

```
state = State.TAP;   // play taps
```

Adding the Drums to the Game

Different ways to add all of the drums in the list to the game

- Write a for loop, go through the list, add each object
- Dart lets us do it without writing our own loop

Add this after you create the drum list.

```
drums.forEach((drum) => game.add(drum));
```

- The list has a built-in method called forEach().
- It takes a function which it calls over and over, passing a different object each time
- (drum) means that the code that follows is a function that takes one argument and we will call it drum. It's one of the drums in the list
- Since the function is only one line of code, dart has a shortcut using => instead of putting it inside { … }

Play the Drums

If you run the game, you should see the 9 drums on the screen, and each one will play its sound when tapped.

If you are having problems, look at the code in GitHub

https://github.com/shawnlg/flutter_ball/tree/08_many_drums

It works much better on a real phone than an emulator.

Next time, we add our buttons to the drum machine.