Finding the Beat

In which we play a tap tap tapping sound
so you can stay on beat
while recording a drum pattern

Metronome

A metronome is a device that clicks at a certain beat to keep you on rhythm

Drum pattern:

- 64 beats that loop
- A beat lasts 1/10 second
- A loud click at the firt beat of the pattern
- A softer click every 8 beats
- **Click** click click click click click click click **Click** click click click click click click click
- Metronome plays while recording
- Does not play when playing back the pattern

Sounds like a job for some constants

Constants

This will demonstrate the usefulness of constants

We can set up the numbers we just showed, but we can easily tweak them later on by changing the constants.

```
// constants
const LOOP_SIZE = 64;
const BEAT_LENGTH = 0.1;  // seconds for each beat in the loop
const METRONOME_BEATS = 8;  // tick sound this many beats apart
const METRONOME_LOUD_SOUND = 'drum/metro main.wav';
const METRONOME_SOFT_SOUND = 'drum/metro off.wav';
const BUTTON_PRESS_SOUND = 'drum/metro off.wav';  // what to play when button is pressed
```

If you don't have these new sound files in your project, be sure to look in GitHub and copy them from there.

https://github.com/shawnlg/flutter_ball/tree/11_metronome

Instance Variables

When designing any program, it is important to think of the data that the program will use.

Think about the functions the drum machine has to perform, and see how many pieces of data you can come up with.

Here's a hint: One piece of data is the beat number we are currently on

Instance Variables

Our data will be stored in instance variables in the drum controller just like we store data for all the other components we have made.

We need a variable to hold which beat number we are on. When playing or recording a pattern, we loop through the pattern starting from the first beat, playing to the end, and starting over again.

```
int currentBeat;  // which beat of the loop we are in
```

For each beat, we need to know what sound we should play. As we record the pattern, more and more beats will have sounds to play. Since a sound can easily be described by its file name, we can make a list of strings to hold all the sounds.

```
List<String> drumTrack = List(LOOP_SIZE);
```

We need to know which beat our metronome should click on. It will start at 0 (the first beat) and be bumped by 8 so that it clicks every 8 beats.

```
int metroBeat;  // which beat to play the next metronome tick
```

Finally, since we deal with time in our update() method, we need to know the time that the next beat starts. This will be a time in seconds from 1970 which is the time the game uses.

```
double timeNextBeat;  // when we have the next beat
```

We will set the instance variables in the constructor

```
// constructor
DrumController(this.game) : super() {
  currentBeat = 0;
  metroBeat = 0;
  timeNextBeat = game.currentTime() + 1;  // start keeping track of beats soon
}
```

Timer Tick

When dealing with repeating time intervals (like in a drum machine), it is common to have a method that acts like the ticking of a timer.

- The shortest amount of time we deal with is the length of a drum beat
- Our “tick” happens once per drum beat
- Our method will control things like when to play a metronome click, which drum was just pressed for recording, which drum we should play back, and when to start the pattern over

Our timer tick method will be called beat()

```
// called once for every beat
void beat() {

    // increment beat number
    currentBeat++;
    if (currentBeat >= LOOP_SIZE) {
      currentBeat = 0;
    }

  }

}
```

For starters, our beat() method keeps track of the beat number.

Before updating the beat to the new number, it will also decide which clicks of the metronome to play. Add this before the code that updates the beat number.

```
// play metronome if recording
if (state == State.RECORD) {
  if (currentBeat == metroBeat) {
    String sound = metroBeat == 0 ? METRONOME_LOUD_SOUND : METRONOME_SOFT_SOUND;
    Flame.audio.play(sound);

    // set next metronome beat
    metroBeat += METRONOME_BEATS;
    if (metroBeat >= LOOP_SIZE) {
      metroBeat = 0;
    }
  }
```

Notice we only play the metronome if we are recording.

We click only when the beat is the beat for the next click

We decide which click to play. The loud click is always on beat 0 (start of pattern)

Record Button

We need to add some logic when we tap the record button. Add this after the code that sets the state to RECORD when button is tapped.

```
currentBeat = 0;   // start of pattern
timeNextBeat = game.currentTime() + 1;  // start first beat soon
metroBeat = 0;   // start ticking on first beat of loop
```

Every time the record button is tapped, we start the pattern over by setting currentBeat to 0. The next beat is set to start in 1 second which gives the user a little time before the pattern starts. And the metronome's first click will be on the first beat – 0.

This is all that is needed to start the metronome ticking.

The beat() method will take care of the rest, and it keeps playing as long as the state is RECORD.

Until Next Time...

Try running the game. You should hear the metronome beating when you record.

If you are on a slow computer running an Android emulator, the beats may sound uneven.

For best results, use a real phone or an iPhone emulator.

Nothing is actually being recorded yet but you can practice playing the drums while recording with the metronome.

Next time we will finish the drum machine.

Remember, if you are having problems, there is working code here:

https://github.com/shawnlg/flutter_ball/tree/11_metronome

TTFN

Play Button

We need to add some logic when we tap the play button. Add this after the code that sets the state to PLAY when button is tapped.

```
currentBeat = 0;   // start of pattern
timeNextBeat = game.currentTime() + 1;   // start playing soon
```

The play button just starts the pattern over by setting currentBeat to 0. The next beat is set to start in 1 second which gives the user a little time before the pattern starts. There is no metronome, so we don't need to change that variable.

Only one thing is left. Think about what it is and how you might do it.

BRB

Timer Tick

We have to call our timer tick method once per beat. And since we have a variable telling us when the next beat is to be done, it's really easy.

Add this to the very end of update() after the tests for states and buttons is done.

```
// count off the beat
if (game.currentTime() > timeNextBeat) {
  timeNextBeat += BEAT_LENGTH;
  beat();
}
```

Once the timer ticks (we call beat() method) we set the time for the next beat.

To see this operate, you could put a print() at the top of the beat() method to print the beat number. Increase the length of the beat by changing BEAT_LENGTH. You will see the beat number loop over and over through the pattern.