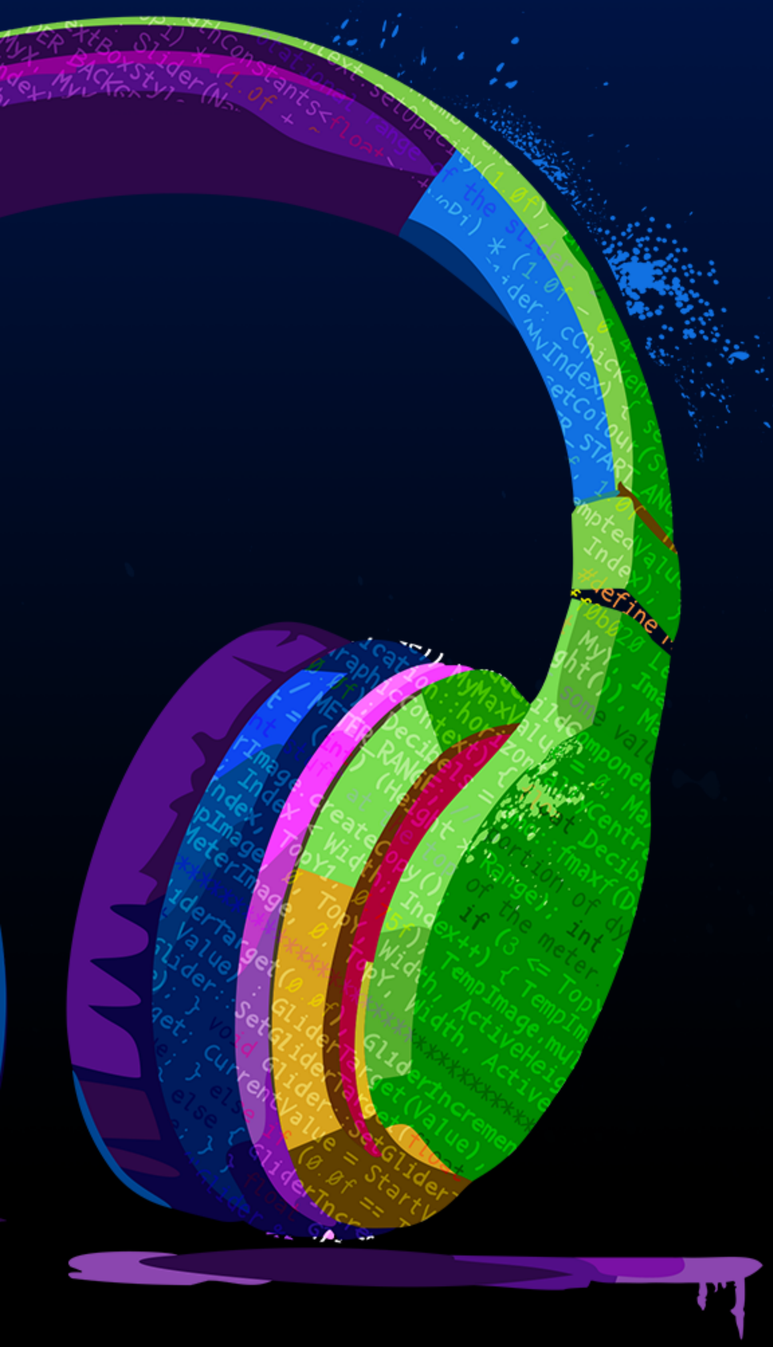


ADC24 Bristol

SNAPSHOT TESTING FOR AUDIO DSP

JOSIP CAVAR

01 Motivation

Speed, quality, cost

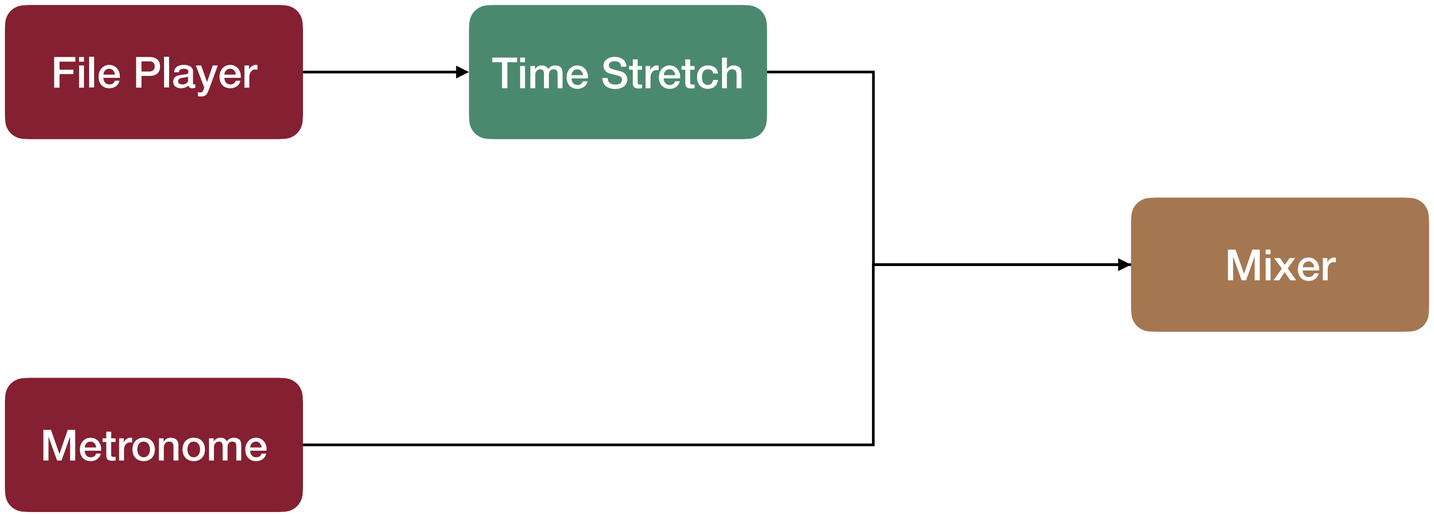
Speed, quality, cost

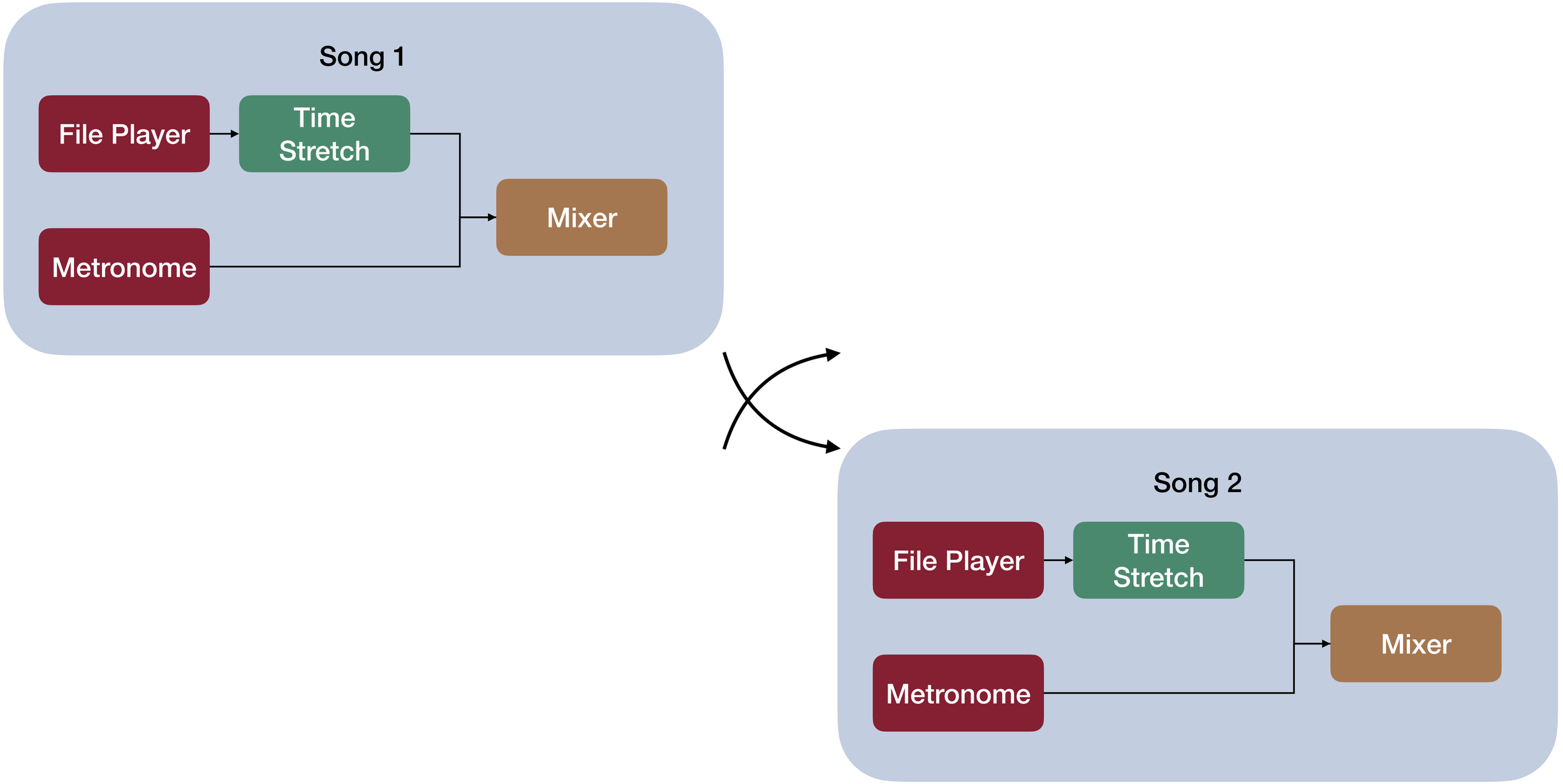
Pick two.

Speed, quality

Speed, quality

Pick one.


File Player
Time Stretch
Mixer
Metronome


Song 1
File Player
Time Stretch
Mixer
Metronome
Song 2
File Player
Time Stretch
Mixer
Metronome

Unit Tests

- Complex setup
- Difficulty in describing expected output
- No iterative feedback

Goals

Unit Tests

- Complex setup
- Difficulty in describing expected output
- No iterative feedback

Goals

Unit Tests

- Complex setup
- Difficulty in describing expected output
- No iterative feedback

Snapshot Tests

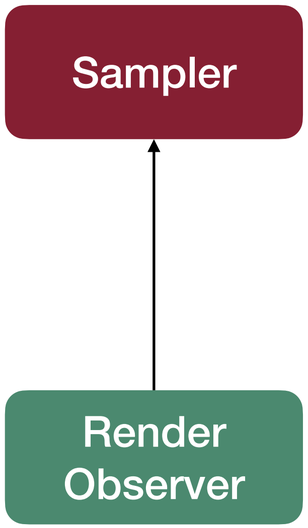
- Easy setup (high-level)
- Using natural visual representation
- Built-in interactivity

02 Writing snapshot tests

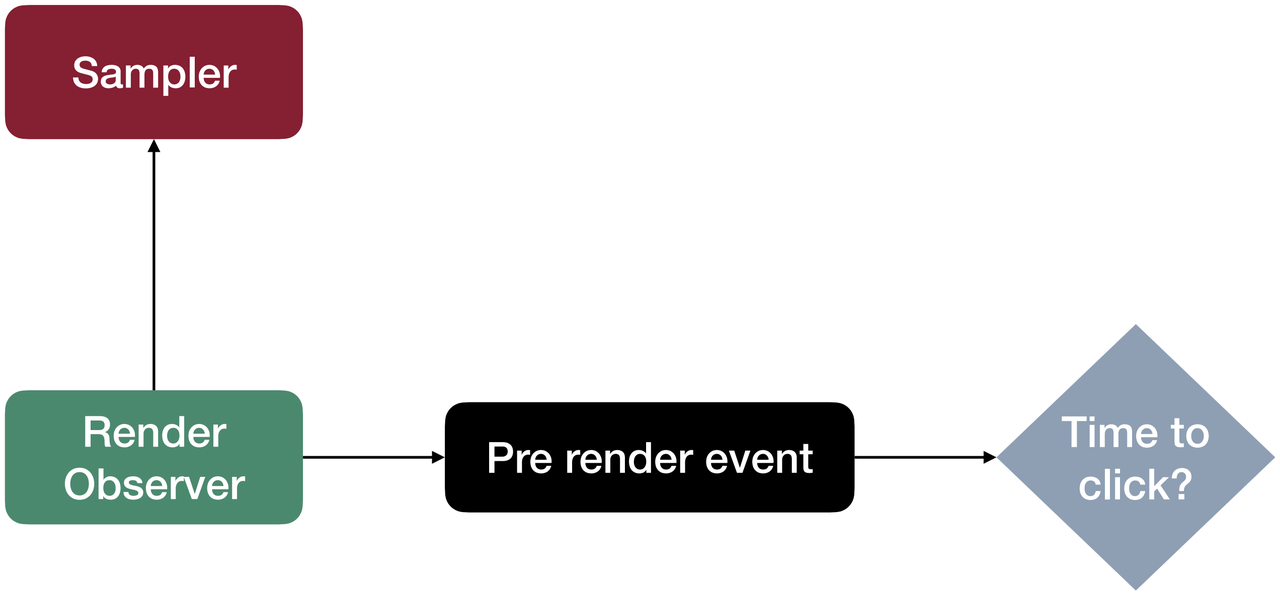
Metronome App

Metronome App

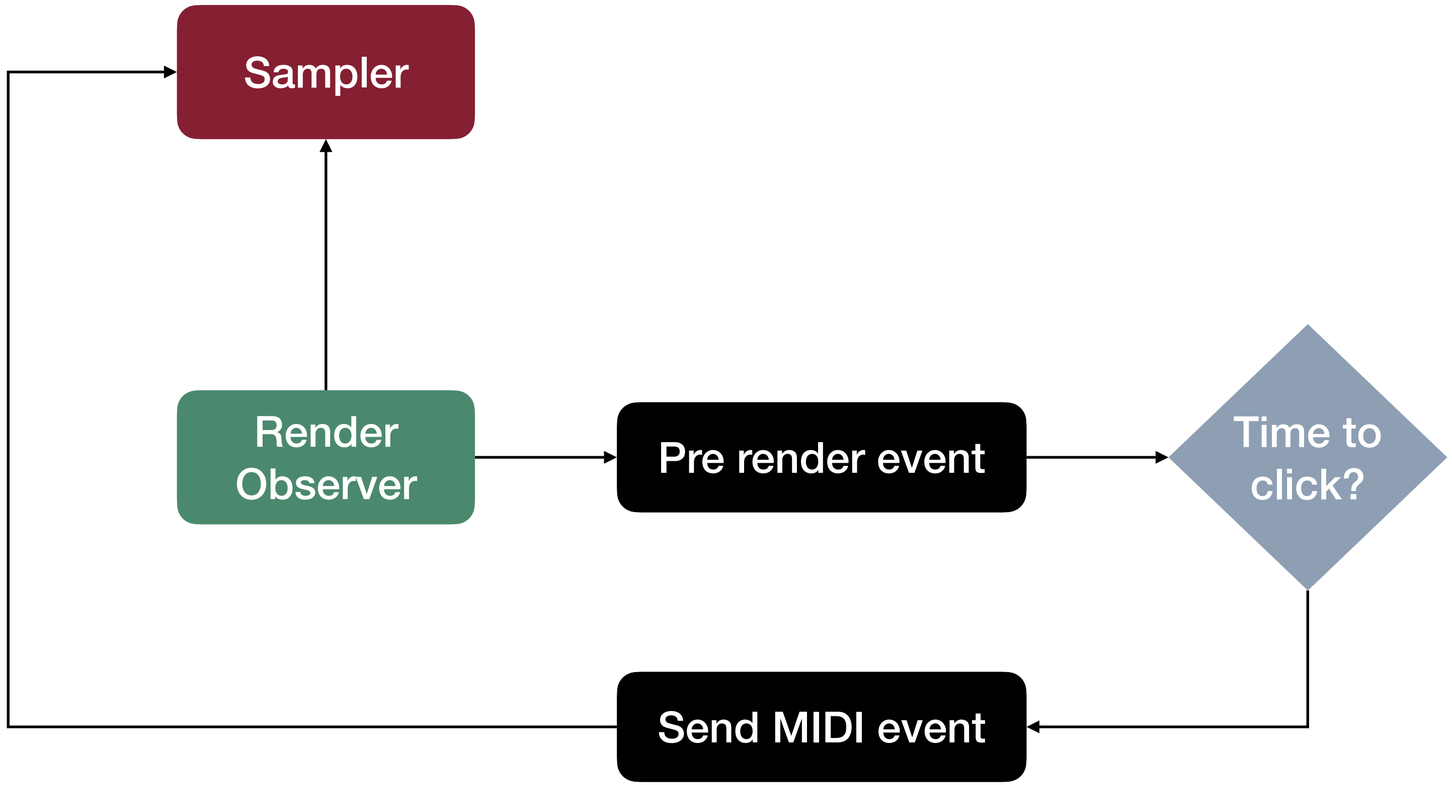
Metronome App



Metronome App



Metronome App



```swift
class Metronome: AVAudioUnitSampler {

    init() {
        super.init()
        let url = Bundle.main.url(
            forResource: "Metronome",
            withExtension: "aupreset"
        )
        try! loadPreset(at: url)
    }
}
```

```
let midiBlock = auAudioUnit.scheduleMIDIEventBlock!
token = auAudioUnit.token(
    byAddingRenderObserver: { flags, timeStamp, frameCount, bus in
        guard flags == .unitRenderAction_PreRender else { return }
        process(
            timeStamp: timeStamp.pointee,
            frameCount: frameCount,
            scheduleMIDI: midiBlock
        )
    }
)
```

```
let midiBlock = auAudioUnit.scheduleMIDIEventBlock!
token = auAudioUnit.token(
    byAddingRenderObserver: { flags, timeStamp, frameCount, bus in
        guard flags == .unitRenderAction_PreRender else { return }
        process(
            timeStamp: timeStamp.pointee,
            frameCount: frameCount,
            scheduleMIDI: midiBlock
        )
    }
)
```

```
let midiBlock = auAudioUnit.scheduleMIDIEventBlock!
token = auAudioUnit.token(
    byAddingRenderObserver: { flags, timeStamp, frameCount, bus in
        guard flags == .unitRenderAction_PreRender else { return }
        process(
            timeStamp: timeStamp.pointee,
            frameCount: frameCount,
            scheduleMIDI: midiBlock
        )
    }
)
```

```
let midiBlock = auAudioUnit.scheduleMIDIEventBlock!
token = auAudioUnit.token(
    byAddingRenderObserver: { flags, timeStamp, frameCount, bus in
        guard flags == .unitRenderAction_PreRender else { return }
        process(
            timeStamp: timeStamp.pointee,
            frameCount: frameCount,
            scheduleMIDI: midiBlock
        )
    }
)
```

```swift
func process(timeStamp: AudioTimeStamp, frameCount: AUAudioFrameCount) {
    let bufferStart = timeStamp.mSampleTime
    let bufferEnd = bufferStart + frameCount

    for sampleTime in bufferStart...bufferEnd {
        if sampleTime.isMultiple(of: sampleRate) {
            let position = sampleTime - bufferStart
            scheduleMIDI(position)
        }
    }
}
```

```
func process(timeStamp: AudioTimeStamp, frameCount: AUAudioFrameCount) {
    let bufferStart = timeStamp.mSampleTime
    let bufferEnd = bufferStart + frameCount

    for sampleTime in bufferStart...bufferEnd {
        if sampleTime.isMultiple(of: sampleRate) {
            let position = sampleTime - bufferStart
            scheduleMIDI(position)
        }
    }
}
```

```
func process(timeStamp: AudioTimeStamp, frameCount: AUAudioFrameCount) {
    let bufferStart = timeStamp.mSampleTime
    let bufferEnd = bufferStart + frameCount

    for sampleTime in bufferStart...bufferEnd {
        if sampleTime.isMultiple(of: sampleRate) {
            let position = sampleTime - bufferStart
            scheduleMIDI(position)
        }
    }
}
```

```swift
func process(timeStamp: AudioTimeStamp, frameCount: AUAudioFrameCount) {
    let bufferStart = timeStamp.mSampleTime
    let bufferEnd = bufferStart + frameCount

    for sampleTime in bufferStart...bufferEnd {
        if sampleTime.isMultiple(of: sampleRate) {
            let position = sampleTime - bufferStart
            scheduleMIDI(position)
        }
    }
}
```

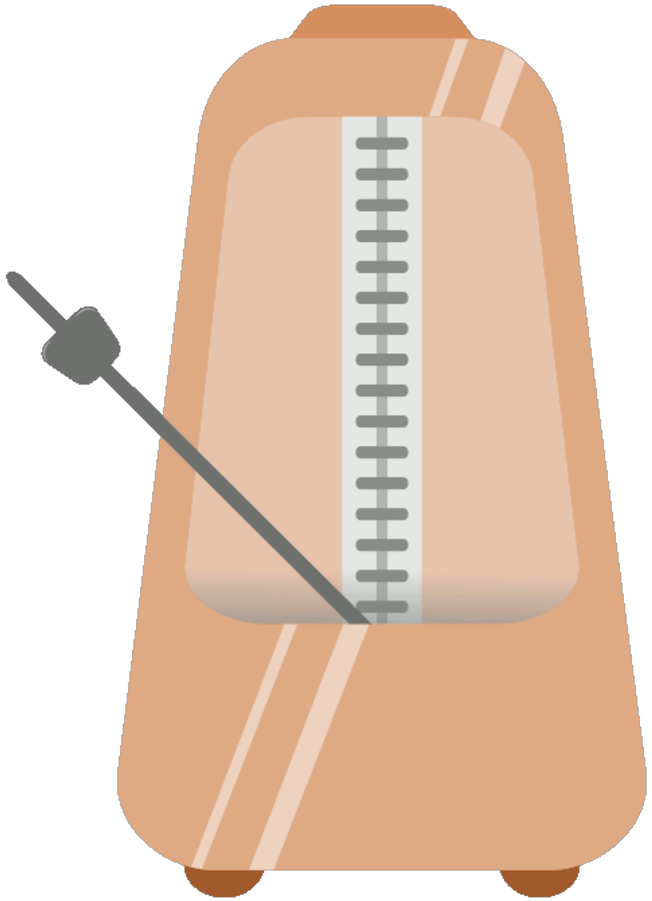
```swift
func process(timeStamp: AudioTimeStamp, frameCount: AUAudioFrameCount) {
    let bufferStart = timeStamp.mSampleTime
    let bufferEnd = bufferStart + frameCount

    for sampleTime in bufferStart...bufferEnd {
        if sampleTime.isMultiple(of: sampleRate) {
            let position = sampleTime - bufferStart
            scheduleMIDI(position)
        }
    }
}
```

```swift
let engine = AVAudioEngine()
let metronome = Metronome()
engine.attach(metronome)
engine.connect(metronome, to: engine.mainMixerNode)
engine.start()
```

```
let engine = AVAudioEngine()
let metronome = Metronome()
engine.attach(metronome)
engine.connect(metronome, to: engine.mainMixerNode)
engine.start()
```

```
let engine = AVAudioEngine()
let metronome = Metronome()
engine.attach(metronome)
engine.connect(metronome, to: engine.mainMixerNode)
engine.start()
```

```
let engine = AVAudioEngine()
let metronome = Metronome()
engine.attach(metronome)
engine.connect(metronome, to: engine.mainMixerNode)
engine.start()
```



A bug

- We get a report that there is a bug

A bug

- We get a report that there is a bug
- How do we approach it?

```
class MetronomeTests {
    let engine = AVAudioEngine()
    let format = AVAudioFormat(sampleRate: 44100, channels: 2)

    init() {
        engine.enableManualRenderingMode(.offline, format: format)
    }
}
```

```
class MetronomeTests {
    let engine = AVAudioEngine()
    let format = AVAudioFormat(sampleRate: 44100, channels: 2)

    init() {
        engine.enableManualRenderingMode(.offline, format: format)
    }
}
```

```
class MetronomeTests {
    let engine = AVAudioEngine()
    let format = AVAudioFormat(sampleRate: 44100, channels: 2)

    init() {
        engine.enableManualRenderingMode(.offline, format: format)
    }
}
```

```swift
@Test
func metronomeEverySecond() throws {
    let node = Metronome()
    engine.connect(node, to: engine.mainMixerNode, format: nil)

    let buffer = AVAudioPCMBuffer(format: format, capacity: 10 * sampleRate)
    let bufferSize = 2048

    try engine.render(bufferSize: bufferSize, format: format, to: buffer)
}
```

```swift
@Test
func metronomeEverySecond() throws {
    let node = Metronome()
    engine.connect(node, to: engine.mainMixerNode, format: nil)

    let buffer = AVAudioPCMBuffer(format: format, capacity: 10 * sampleRate)
    let bufferSize = 2048

    try engine.render(bufferSize: bufferSize, format: format, to: buffer)
}
```

```swift
@Test
func metronomeEverySecond() throws {
    let node = Metronome()
    engine.connect(node, to: engine.mainMixerNode, format: nil)

    let buffer = AVAudioPCMBuffer(format: format, capacity: 10 * sampleRate)
    let bufferSize = 2048

    try engine.render(bufferSize: bufferSize, format: format, to: buffer)
}
```

```swift
@Test
func metronomeEverySecond() throws {
    let node = Metronome()
    engine.connect(node, to: engine.mainMixerNode, format: nil)

    let buffer = AVAudioPCMBuffer(format: format, capacity: 10 * sampleRate)
    let bufferSize = 2048

    try engine.render(bufferSize: bufferSize, format: format, to: buffer)
}
```

```swift
@Test
func metronomeEverySecond() throws {
    let node = Metronome()
    engine.connect(node, to: engine.mainMixerNode, format: nil)

    let buffer = AVAudioPCMBuffer(format: format, capacity: 10 * sampleRate)
    let bufferSize = 2048

    try engine.render(bufferSize: bufferSize, format: format, to: buffer)
}
```

Snapshot tests

Snapshot tests

- assertSnapshot(of: Value, as: Strategy)

Snapshot tests

- assertSnapshot(of: Value, as: Strategy)
- Compare against what?

Snapshot tests

- assertSnapshot(of: Value, as: Strategy)
- Compare against what?
- Two modes

Snapshot tests

- assertSnapshot(of: Value, as: Strategy)
- Compare against what?
- Two modes
  - Interactive (recording)

Snapshot tests

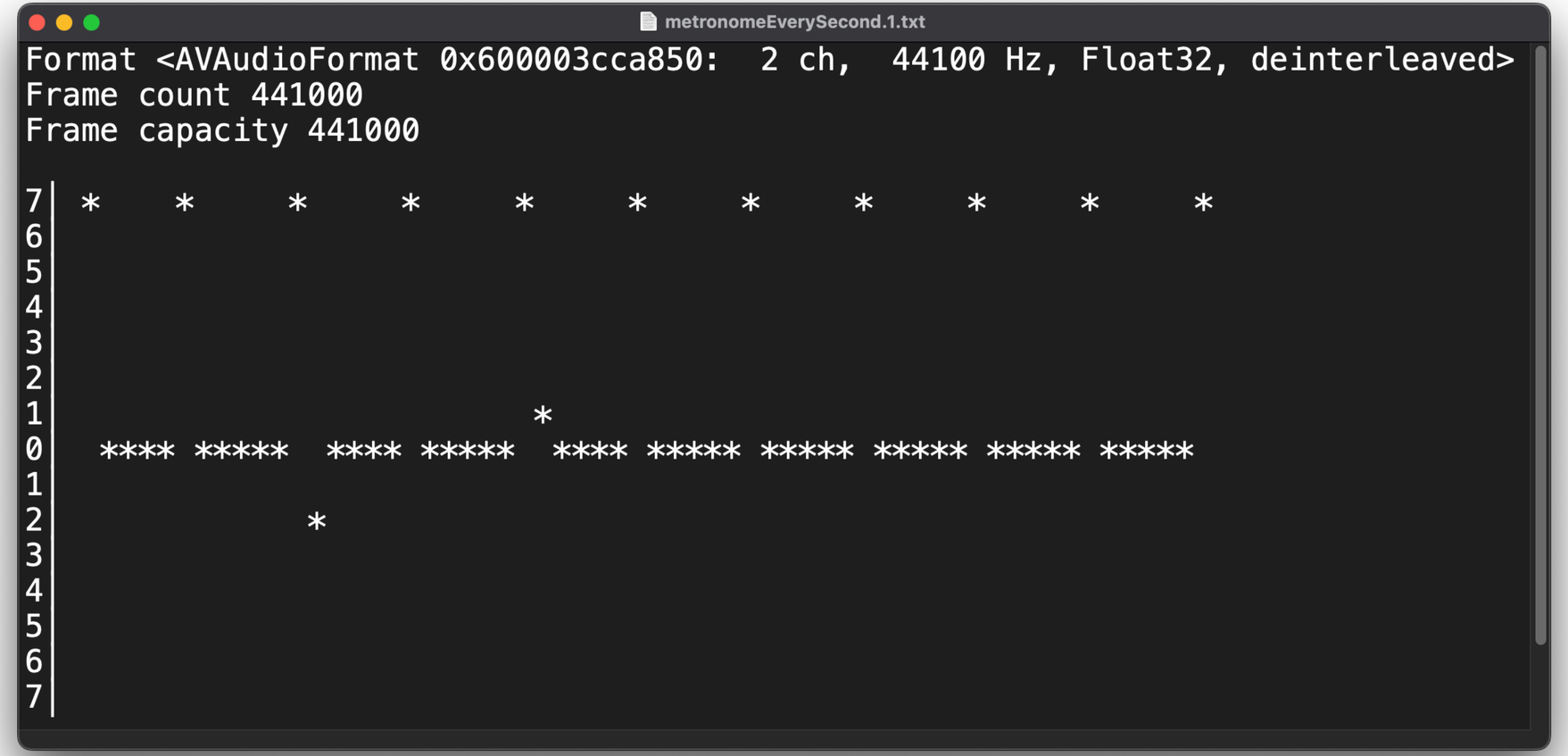
- assertSnapshot(of: Value, as: Strategy)
- Compare against what?
- Two modes
  - Interactive (recording)
  - Non-interactive (asserting)

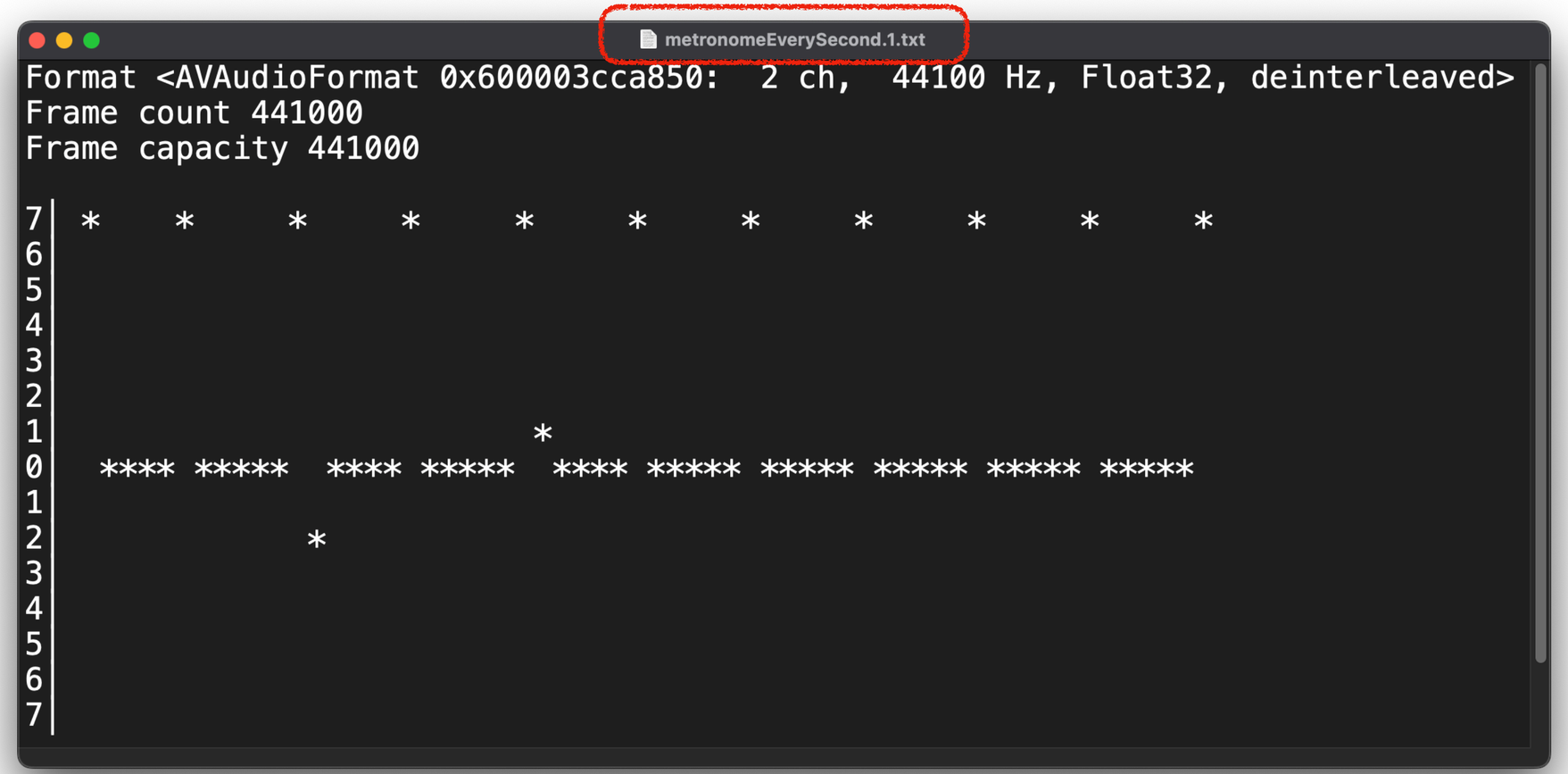
```swift
@Test(.snapshots(record: true))
func metronomeEverySecond() {
    // ...

    assertSnapshot(of: buffer, as: .bufferText(width: 120, height: 15))
}
```

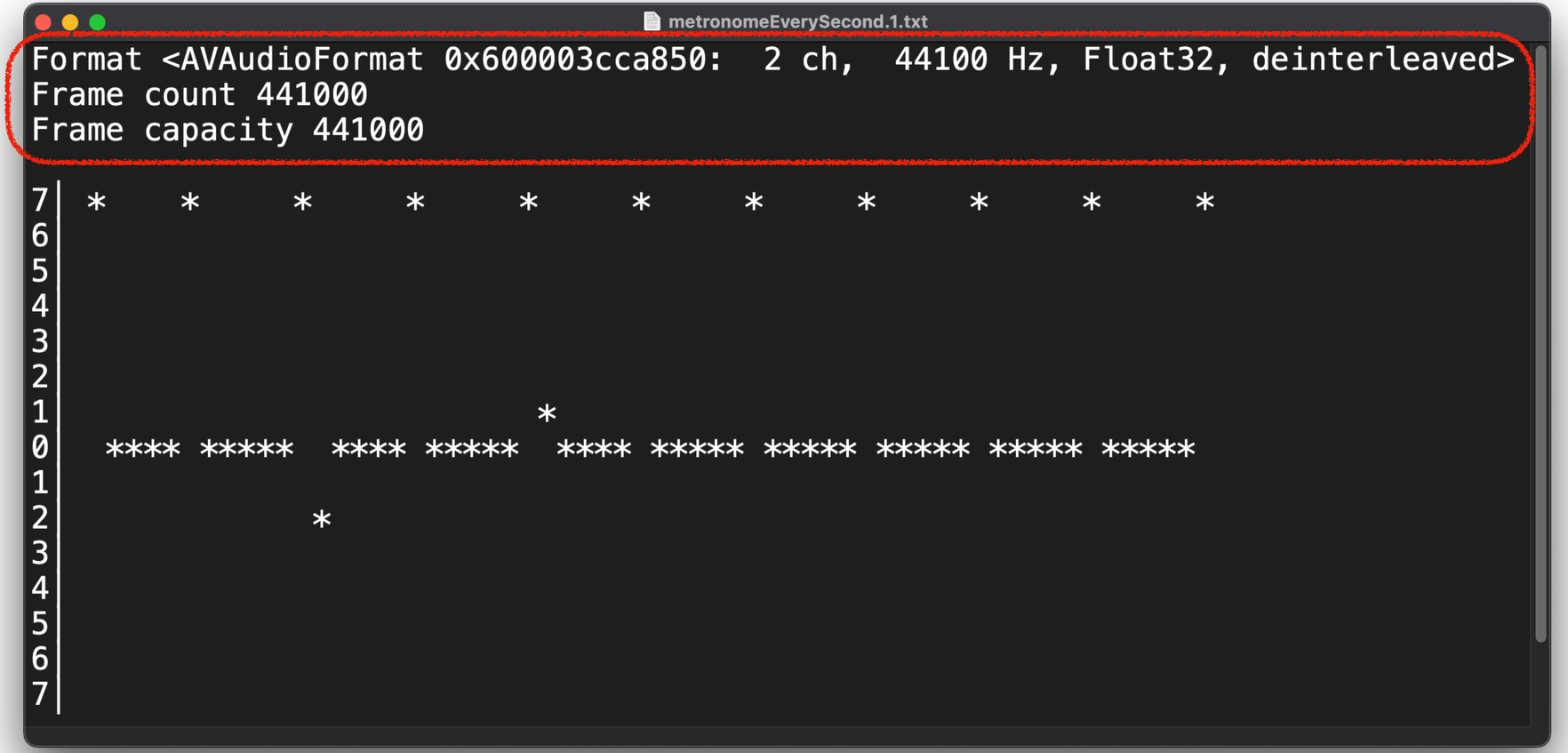
```swift
@Test(.snapshots(record: true))
func metronomeEverySecond() {
    // ...

    assertSnapshot(of: buffer, as: .bufferText(width: 120, height: 15))
}
```

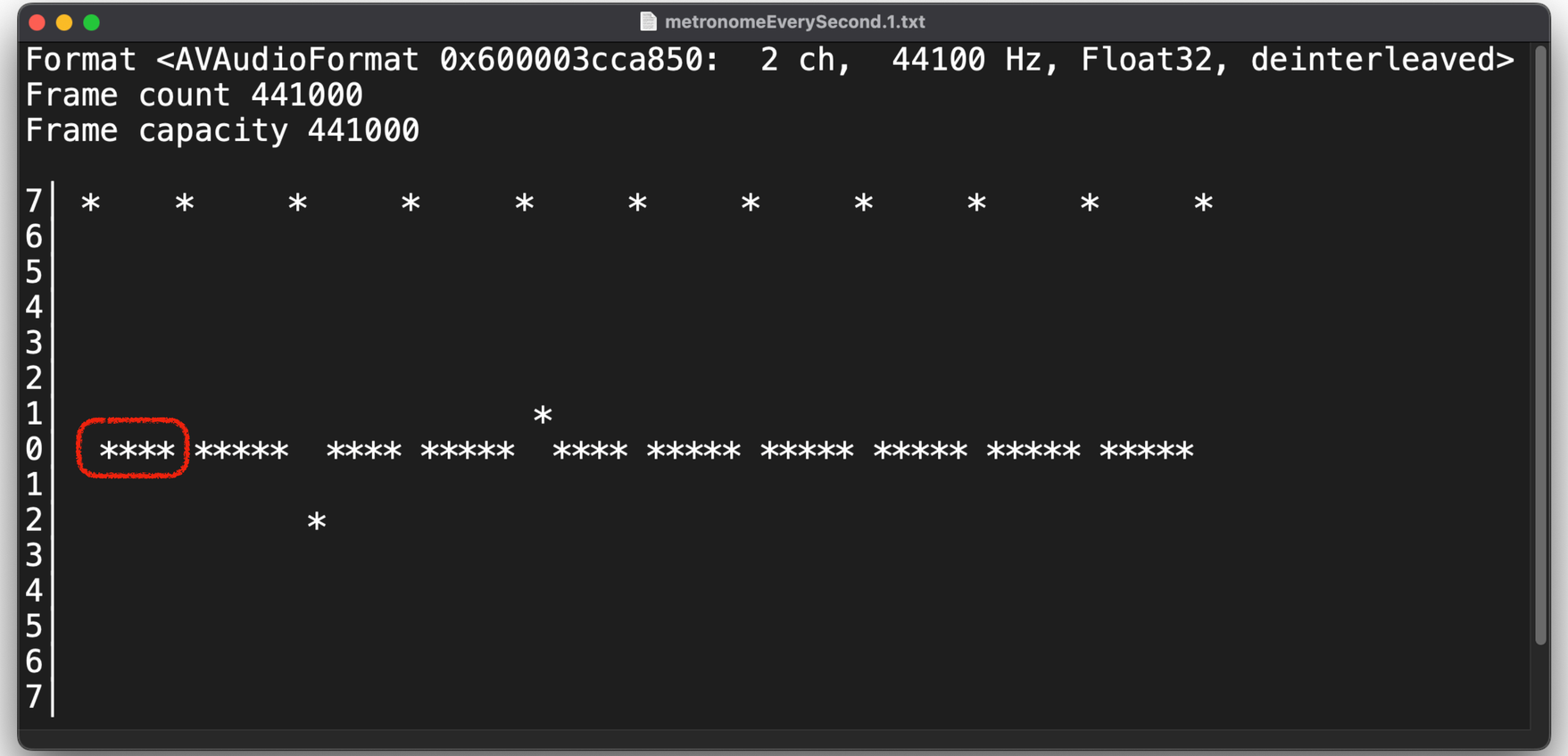
```swift
@Test(.snapshots(record: true))
func metronomeEverySecond() {
    // ...

    assertSnapshot(of: buffer, as: .bufferText(width: 120, height: 15))
}
```

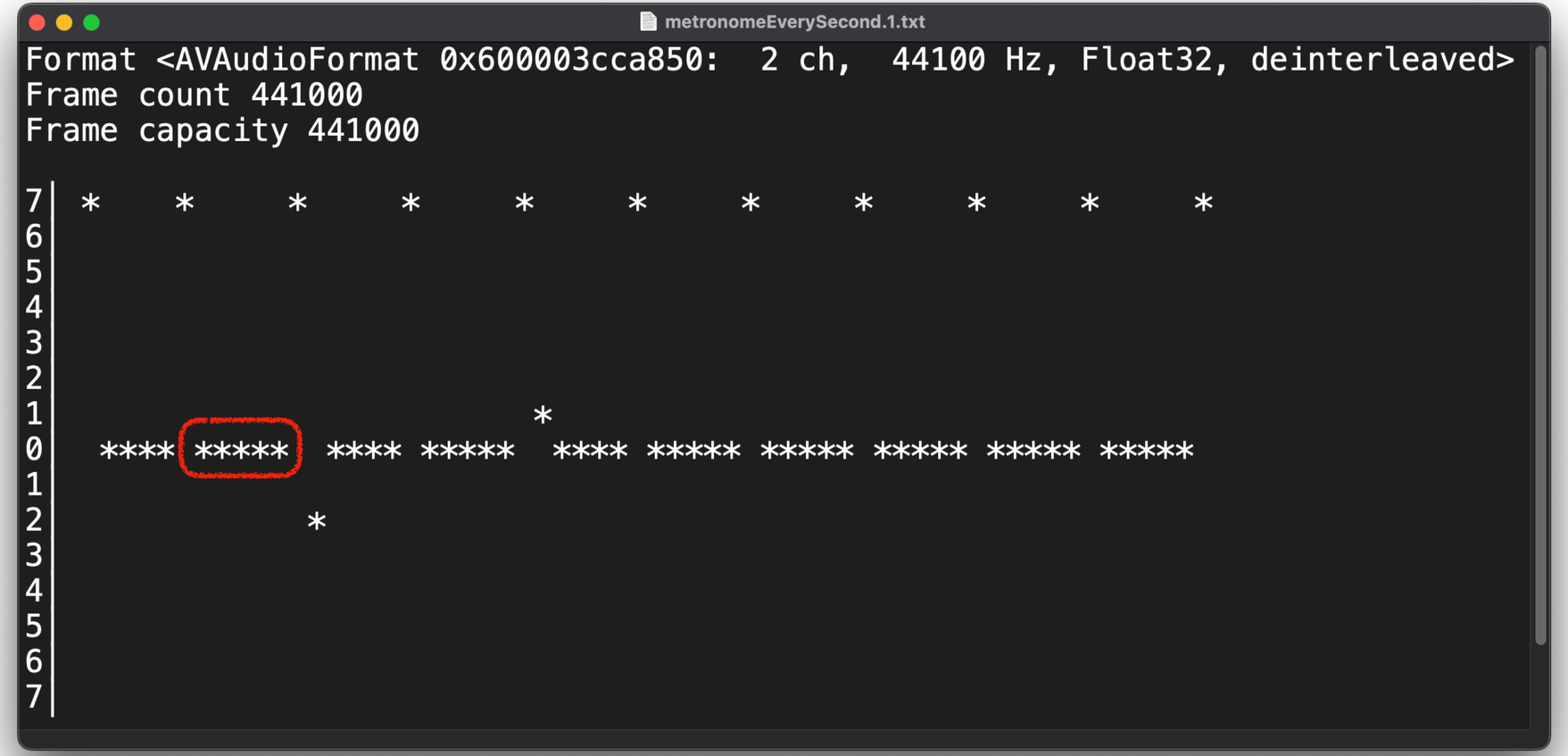
```swift
@Test(.snapshots(record: true))
func metronomeEverySecond() {
    // ...

    assertSnapshot(of: buffer, as: .bufferText(width: 120, height: 15))
}
```


metronomeEverySecond.1.txt
Format <AVAudioFormat 0x600003cca850: 2 ch, 44100 Hz, Float32, deinterleaved>
Frame count 441000
Frame capacity 441000
7 | *   *   *   *   *   *   *   *   *   *   *
6 |
5 |
4 |
3 |
2 |
1 | *
0 | **** ***** **** ***** **** ***** ***** ***** ***** *****
1 |
2 | *
3 |
4 |
5 |
6 |
7 |


metronomeEverySecond.1.txt
Format <AVAudioFormat 0x600003cca850: 2 ch, 44100 Hz, Float32, deinterleaved>
Frame count 441000
Frame capacity 441000
7 | *   *   *   *   *   *   *   *   *   *   *
6 |
5 |
4 |
3 |
2 |
1 | *
0 | **** ***** **** ***** **** ***** ***** ***** ***** *****
1 |
2 | *
3 |
4 |
5 |
6 |
7 |

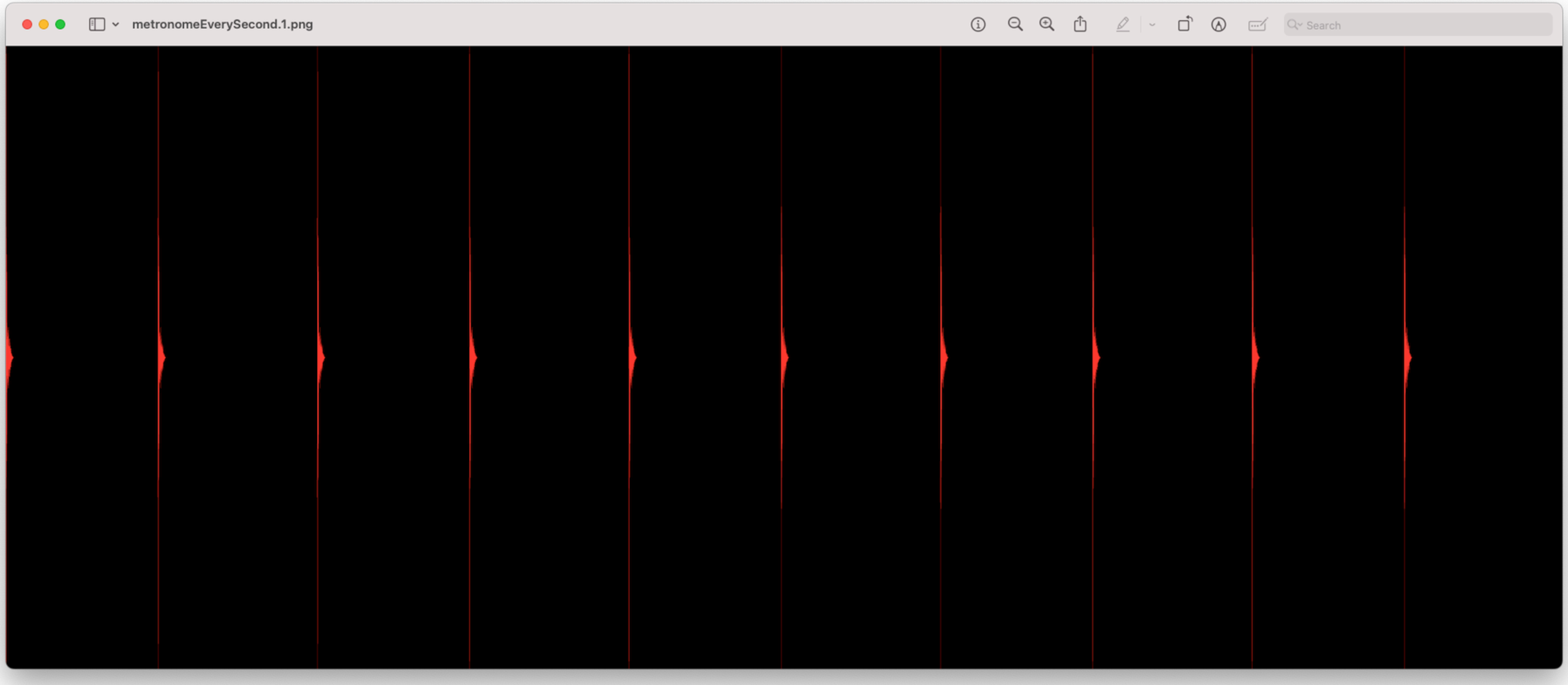

metronomeEverySecond.1.txt
Format <AVAudioFormat 0x600003cca850: 2 ch, 44100 Hz, Float32, deinterleaved>
Frame count 441000
Frame capacity 441000
7 | *   *   *   *   *   *   *   *   *   *   *
6 |
5 |
4 |
3 |
2 |
1 | *
0 | **** ***** **** ***** **** ***** ***** ***** ***** *****
1 |
2 | *
3 |
4 |
5 |
6 |
7 |

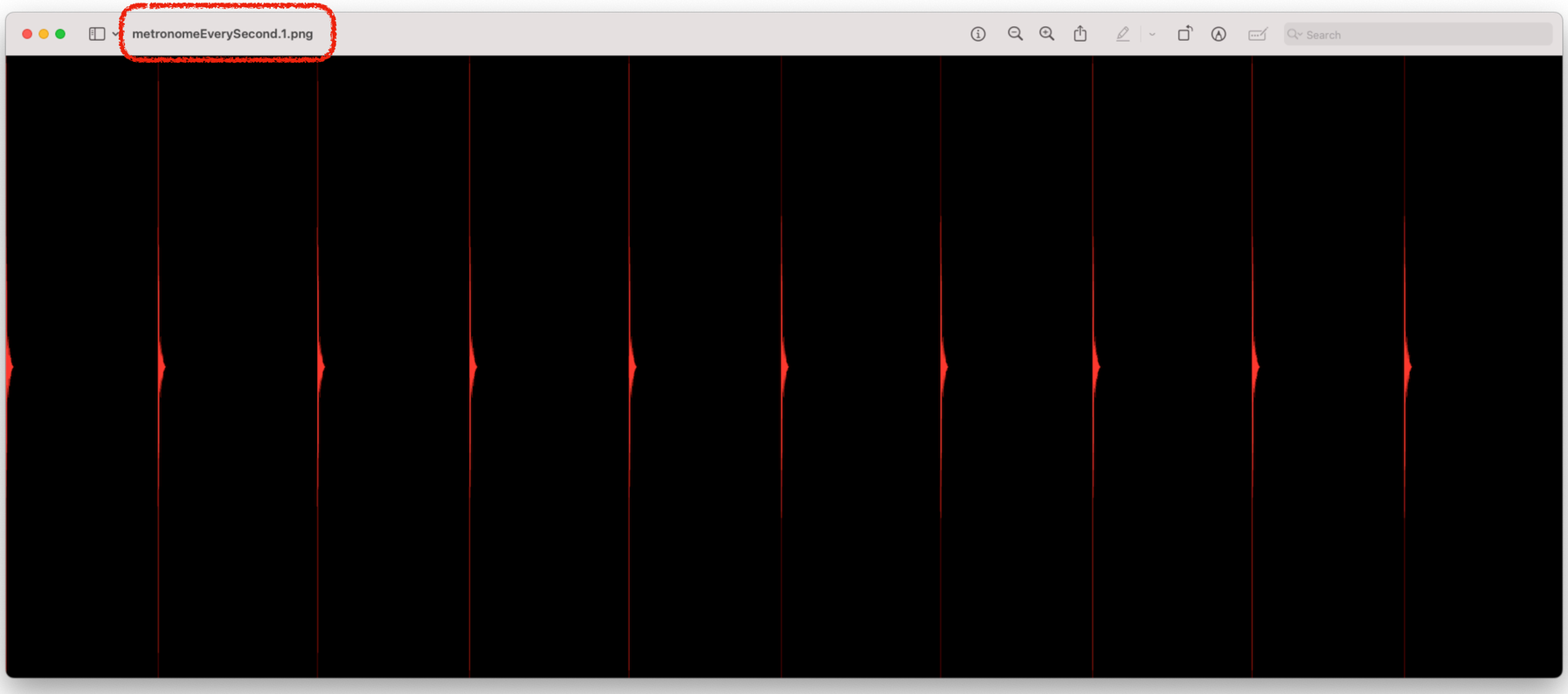

metronomeEverySecond.1.txt
Format <AVAudioFormat 0x600003cca850: 2 ch, 44100 Hz, Float32, deinterleaved>
Frame count 441000
Frame capacity 441000
7
6
5
4
3
2
1
0
1
2
3
4
5
6
7


metronomeEverySecond.1.txt
Format <AVAudioFormat 0x600003cca850: 2 ch, 44100 Hz, Float32, deinterleaved>
Frame count 441000
Frame capacity 441000
7 * * * * * * * * * * *
6
5
4
3
2
1 *
0 **** ***** **** ***** **** ***** ***** ***** ***** *****
1
2 *
3
4
5
6
7

```swift
@Test(.snapshots(record: true))
func metronomeEverySecond() {
    // ...

    assertSnapshot(of: buffer, as: .bufferImage(width: 10000, height: 500))
}
```

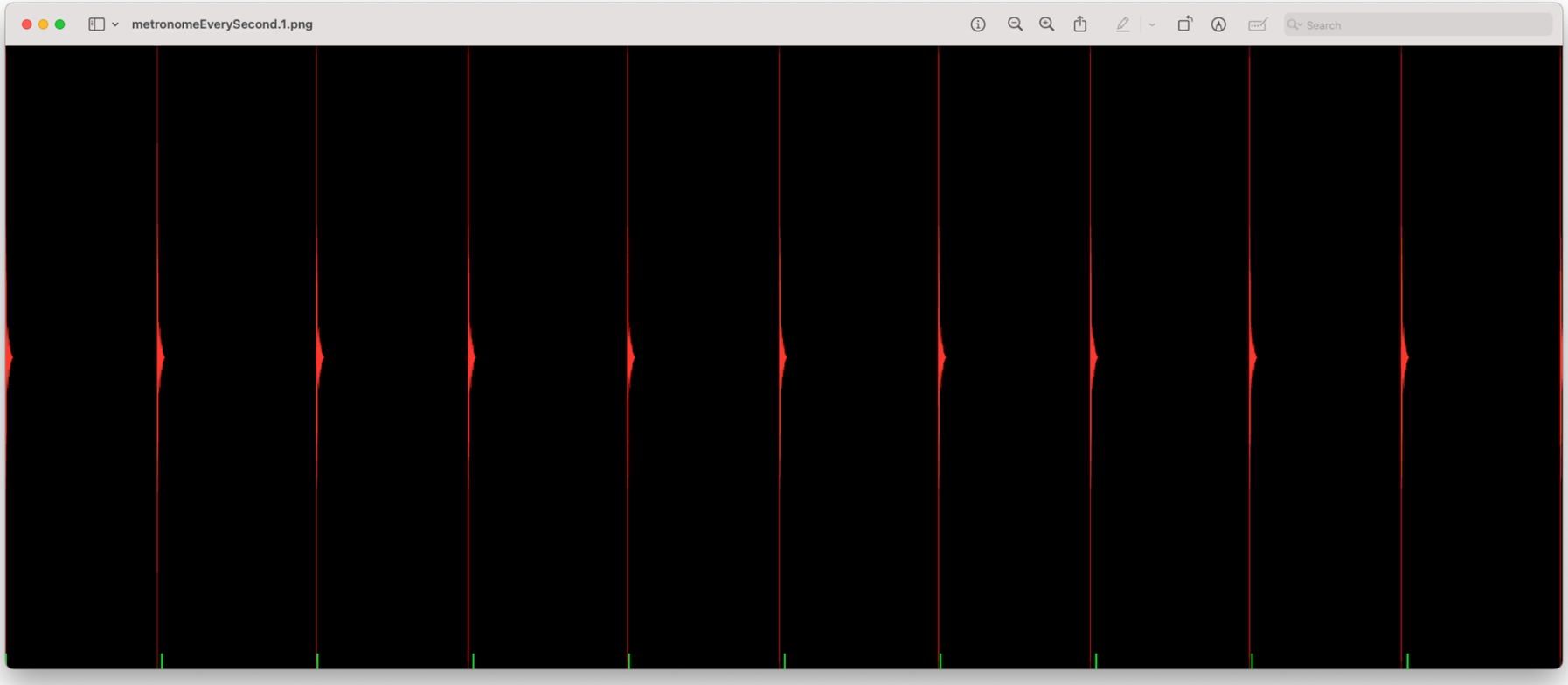
```swift
@Test(.snapshots(record: true))
func metronomeEverySecond() {
    // ...

    assertSnapshot(of: buffer, as: .bufferImage(width: 10000, height: 500))
}
```

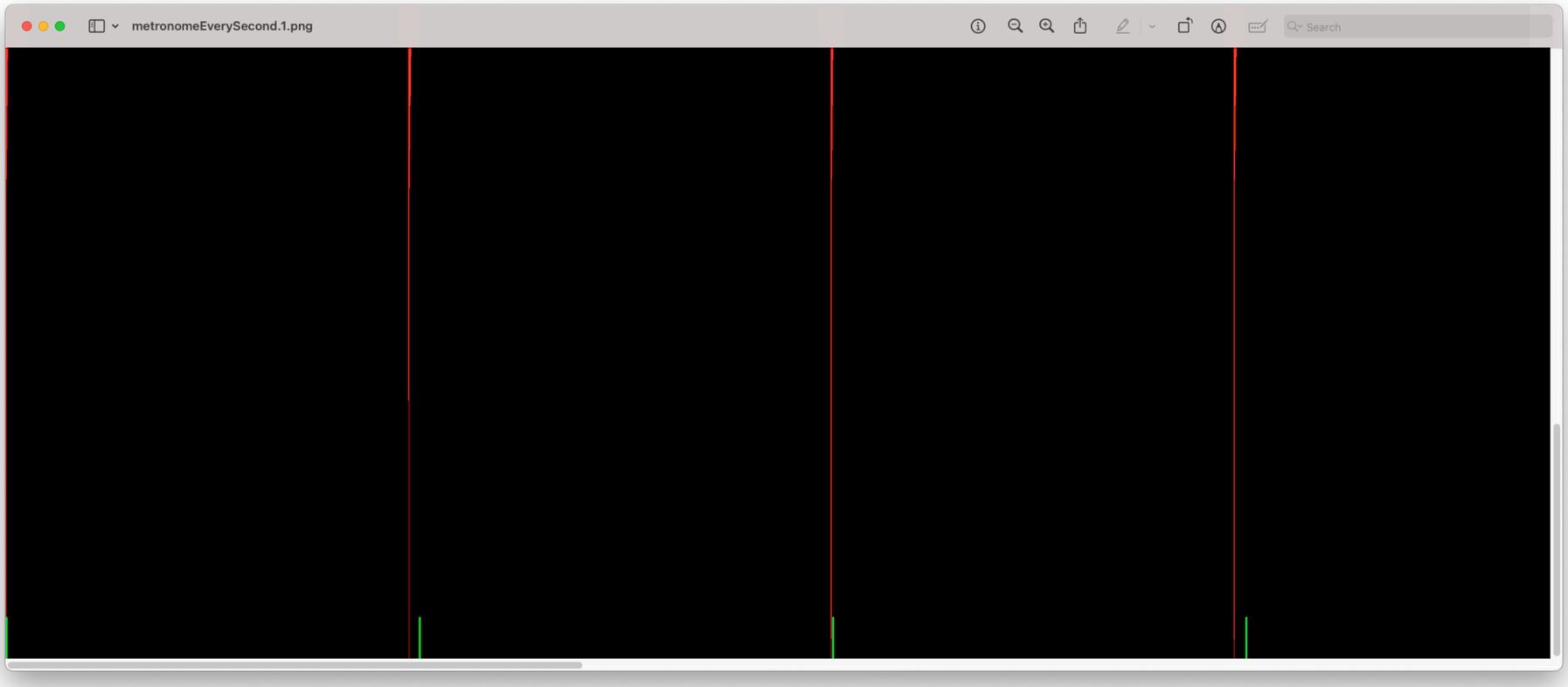

metronomeEverySecond.1.png
Search


metronomeEverySecond.1.png
Search

```swift
@Test(.snapshots(record: true))
func metronomeEverySecond() {
    // ...

    assertSnapshot(
        of: buffer,
        as: .bufferImage(width: 10000, height: 500, overlay: TimelineView())
    )
}
```

```swift
@Test(.snapshots(record: true))
func metronomeEverySecond() {
    // ...

    assertSnapshot(
        of: buffer,
        as: .bufferImage(width: 10000, height: 500, overlay: TimelineView())
    )
}
```


metronomeEverySecond.1.png
Search


metronomeEverySecond.1.png
Search

Recap

- Automated test that reproduces the issue
- Visually verify the correctness of the output
- We can iterate on the fix

AUScheduleMIDIEventBlock

Discussion

eventSampleTime

The sample time at which the MIDI event is to occur. When scheduling parameters during the render cycle, this time can be set to the `AUEventSampleTimeImmediate` value plus an optional buffer offset, in which case the event is scheduled at that position in the current render cycle.

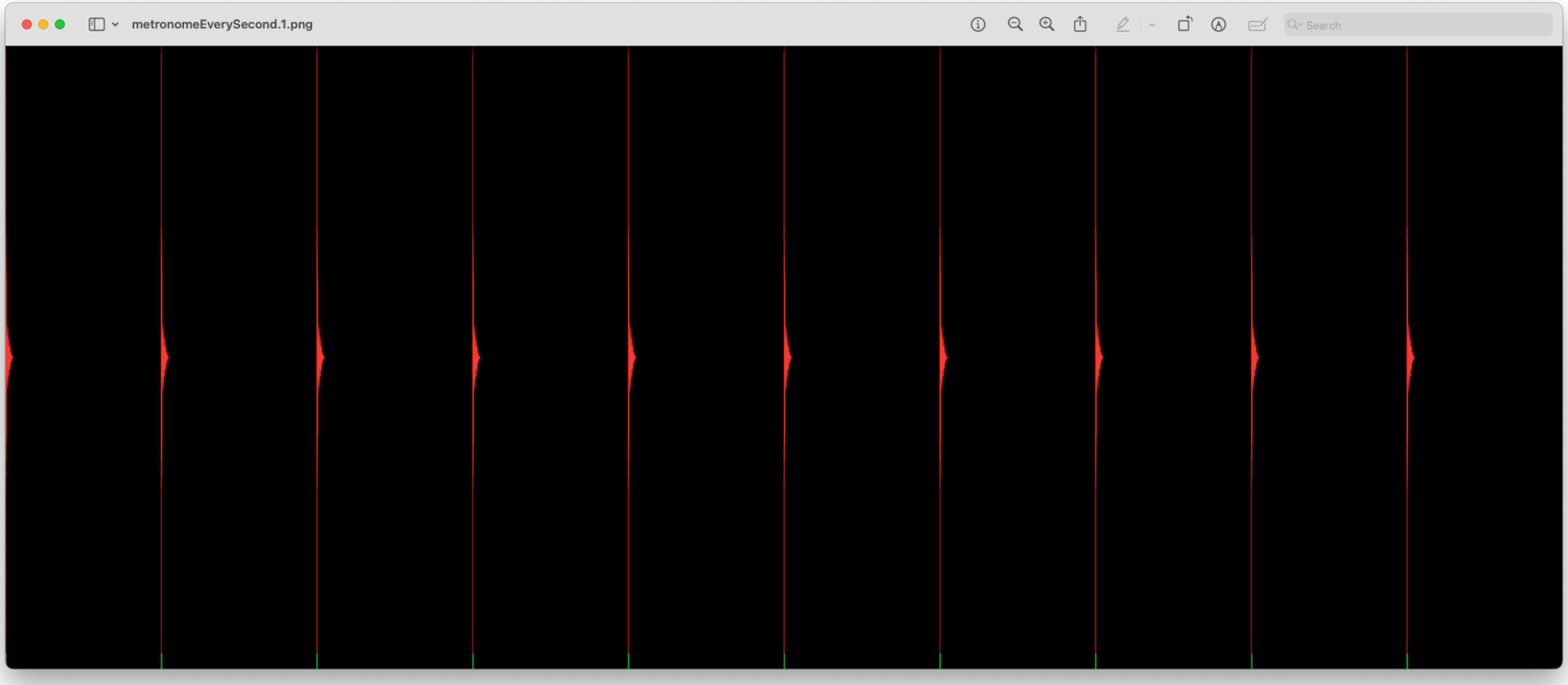
AUScheduleMIDIEventBlock

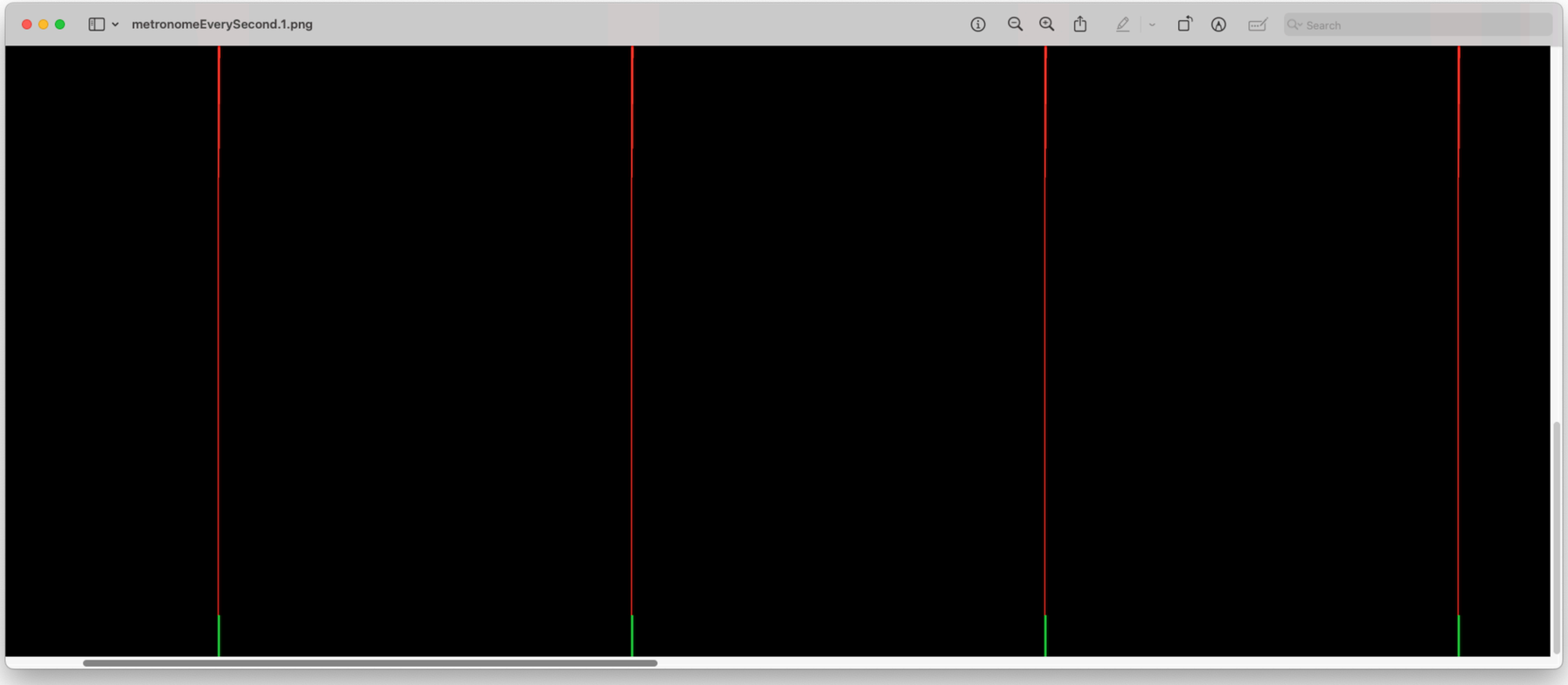
Discussion

eventSampleTime

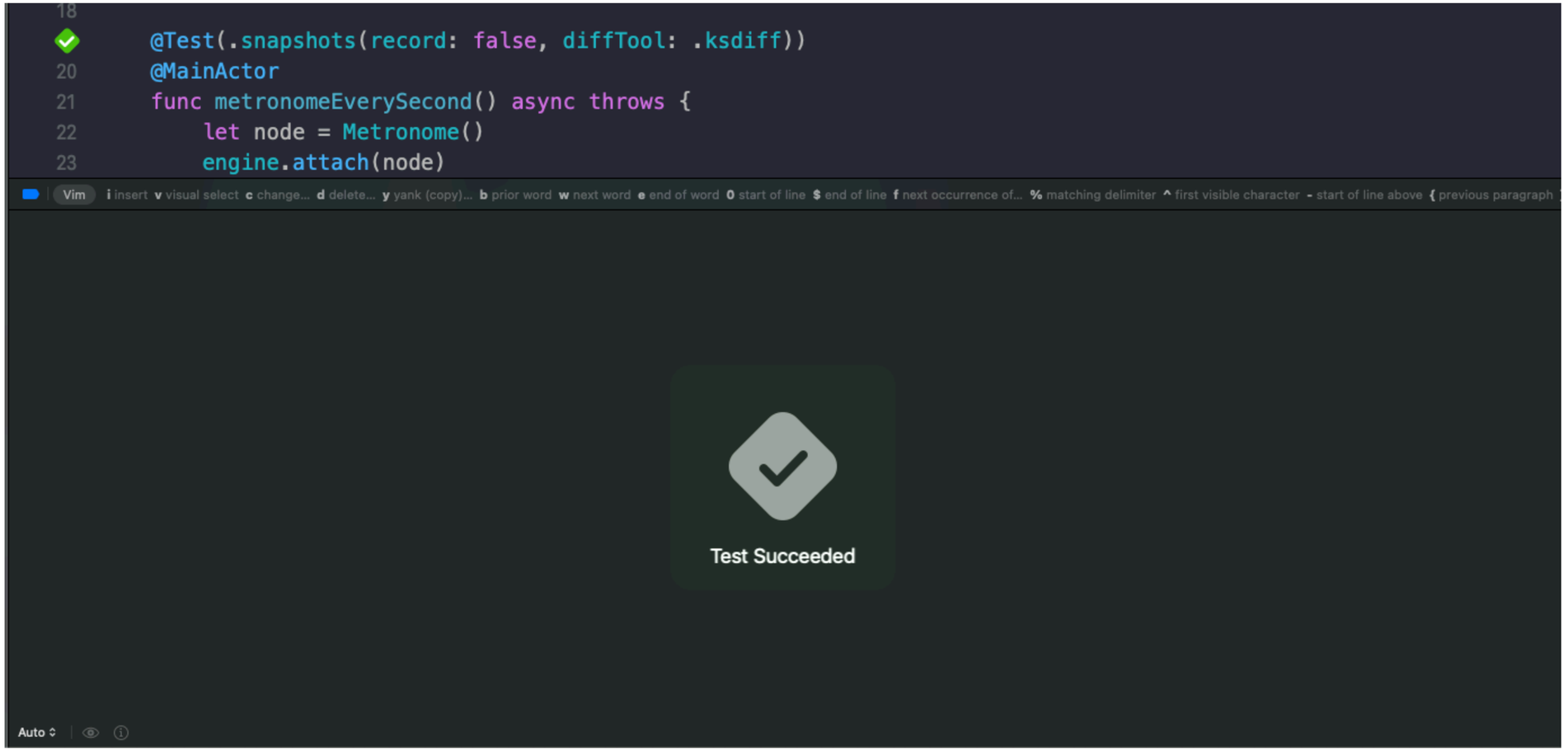
The sample time at which the MIDI event is to occur. When scheduling parameters during the render cycle, this time can be set to the `AUEventSampleTimeImmediate` value plus an optional buffer offset, in which case the event is scheduled at that position in the current render cycle.

```
scheduleMIDI(AUEventSampleTimeImmediate + position)
```


metronomeEverySecond.1.png
Search


metronomeEverySecond.1.png
Search

```
@Test(.snapshots(record: false))
```


18
@Test(.snapshots(record: false, diffTool: .ksdiff))
20 @MainActor
21 func metronomeEverySecond() async throws {
22 let node = Metronome()
23 engine.attach(node)
Vim i insert v visual select c change... d delete... y yank (copy)... b prior word w next word e end of word 0 start of line $ end of line f next occurrence of... % matching delimiter ^ first visible character - start of line above { previous paragraph
Test Succeeded
Auto

Fix incorrectly timed MIDI event #1

Edit · Code

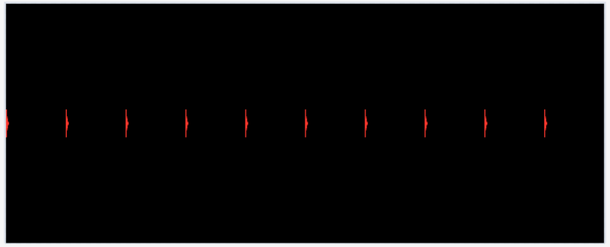
Open · jcavar wants to merge 1 commit into `main` from `fix/sample-time-immediate`

Conversation 0 · Commits 1 · Checks 0 · **Files changed 4** · +59 −1

2 Metronome/Audio/Metronome.swift — Viewed

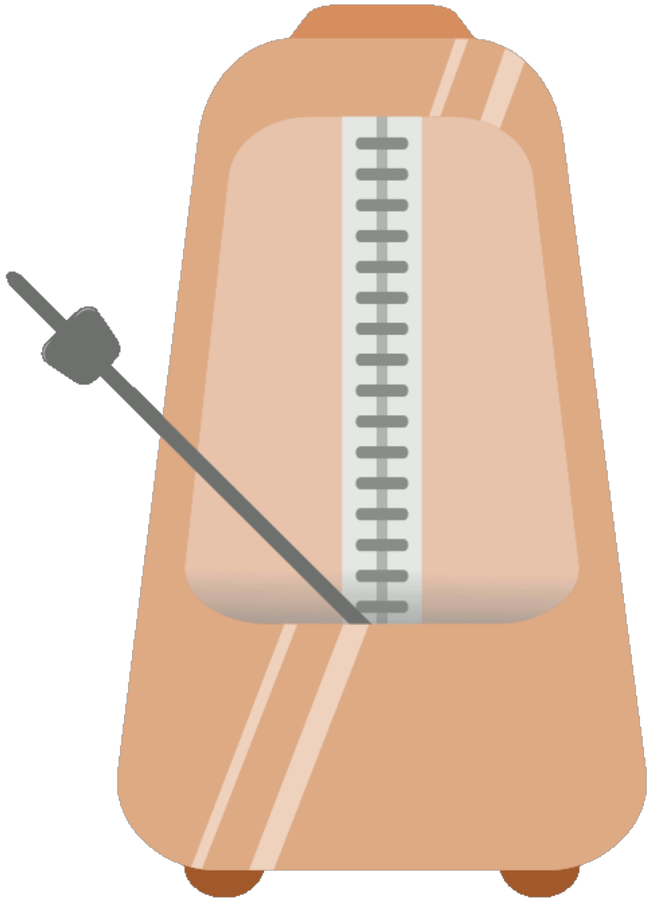
```
@@ -48,7 +48,7 @@ private func process(
48      for sampleTime in bufferStart...bufferEnd {
49          if sampleTime.isMultiple(of: Int64(sampleRate)) {
50              let position = sampleTime - bufferStart
51 -            scheduleMIDI(position)
51 +            scheduleMIDI(AUEventSampleTimeImmediate + position)
52          }
53      }
54  }
```

BIN +2.69 MB MetronomeTests/__Snapshots__/MetronomeTests/metronomeEverySecond.1.png — Viewed



19 MetronomeTests/__Snapshots__/MetronomeTests/metronomeEverySecond.2.txt — Viewed

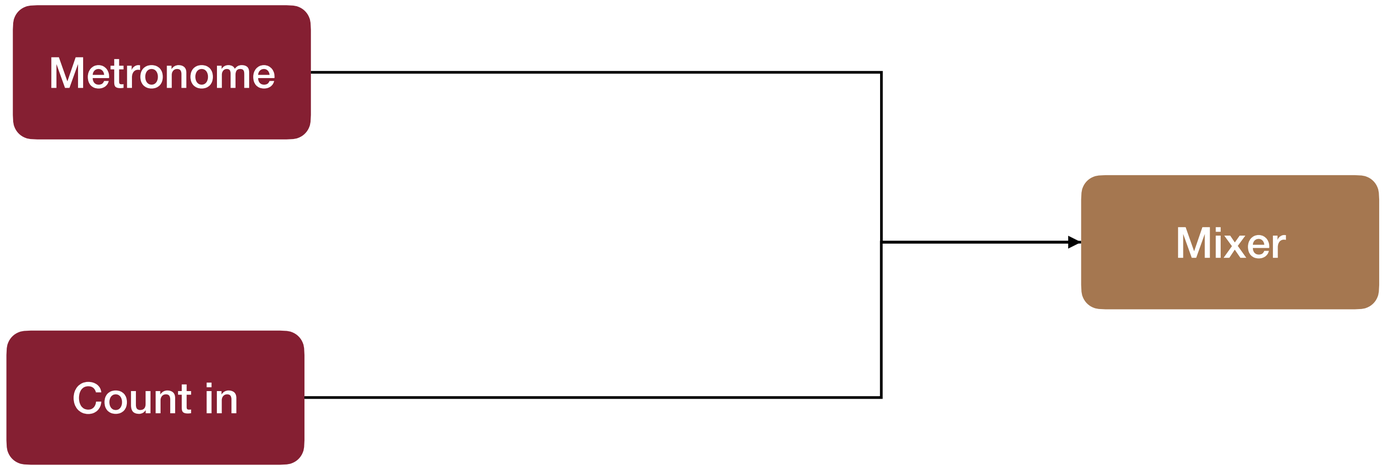
```
@@ -0,0 +1,19 @@
1  + Format 2 ch, 44100.0 Hz, deinterleaved, Float32
2  + Frame count 441000
3  + Frame capacity 441000
4  +
5  + 7|  *     *     *     *     *     *     *     *     *     *
6  + 6|
7  + 5|
8  + 4|
9  + 3|
10 + 2|
11 + 1|
12 + 0|  ***** ***** ***** ***** ***** ***** ***** ***** ***** *****
13 + 1|
```



A new feature

- Count in 🥁

Count in + Metronome



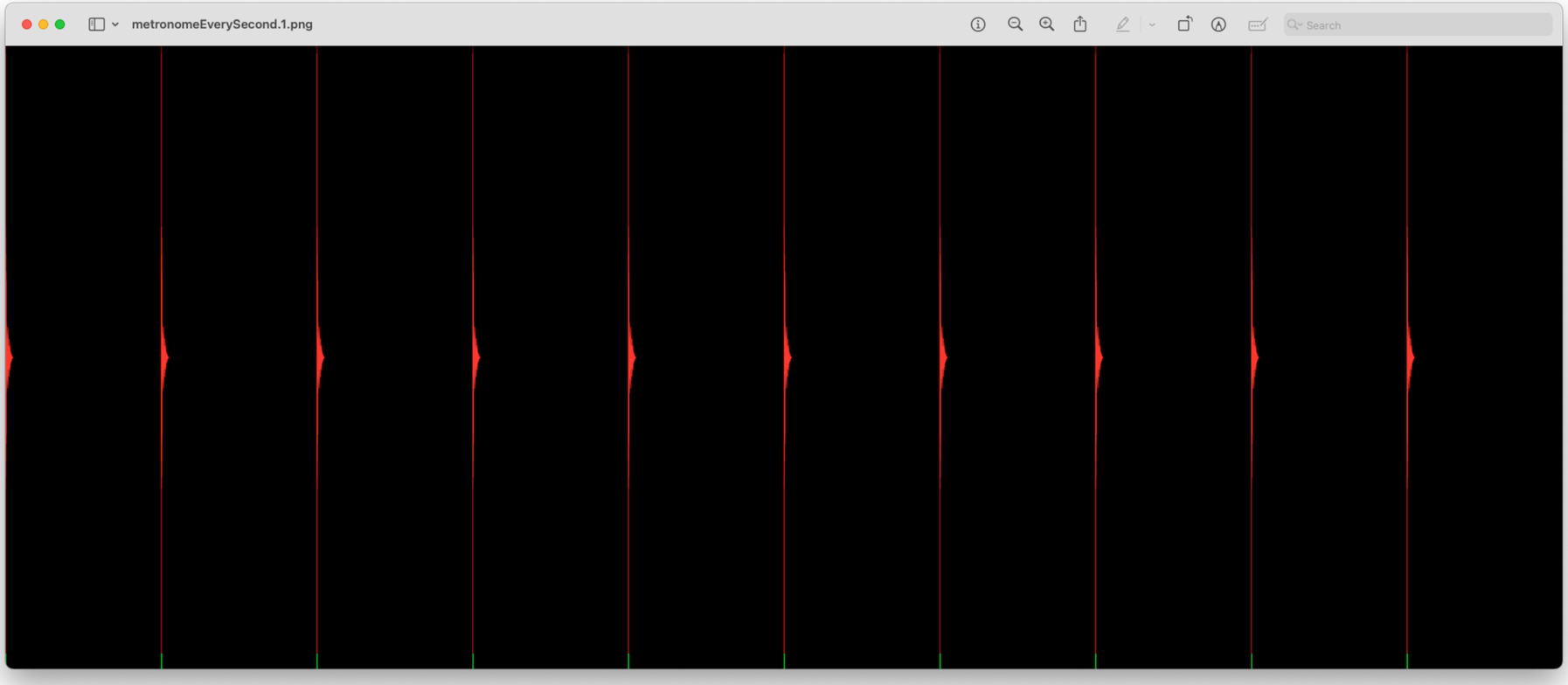
```swift
@Test(.snapshots(record: true))
func metronomeAndCountIn() async throws {
    let metro = Metronome()
    engine.connect(metro, to: engine.mainMixerNode, format: nil)

    let count = CountIn()
    engine.connect(count, to: engine.mainMixerNode, format: nil)

    try engine.render(bufferSize: bufferSize, format: format, to: buffer)

    assertSnapshot(of: buffer, as: .bufferImage(width: 10000, height: 4000))
}
```

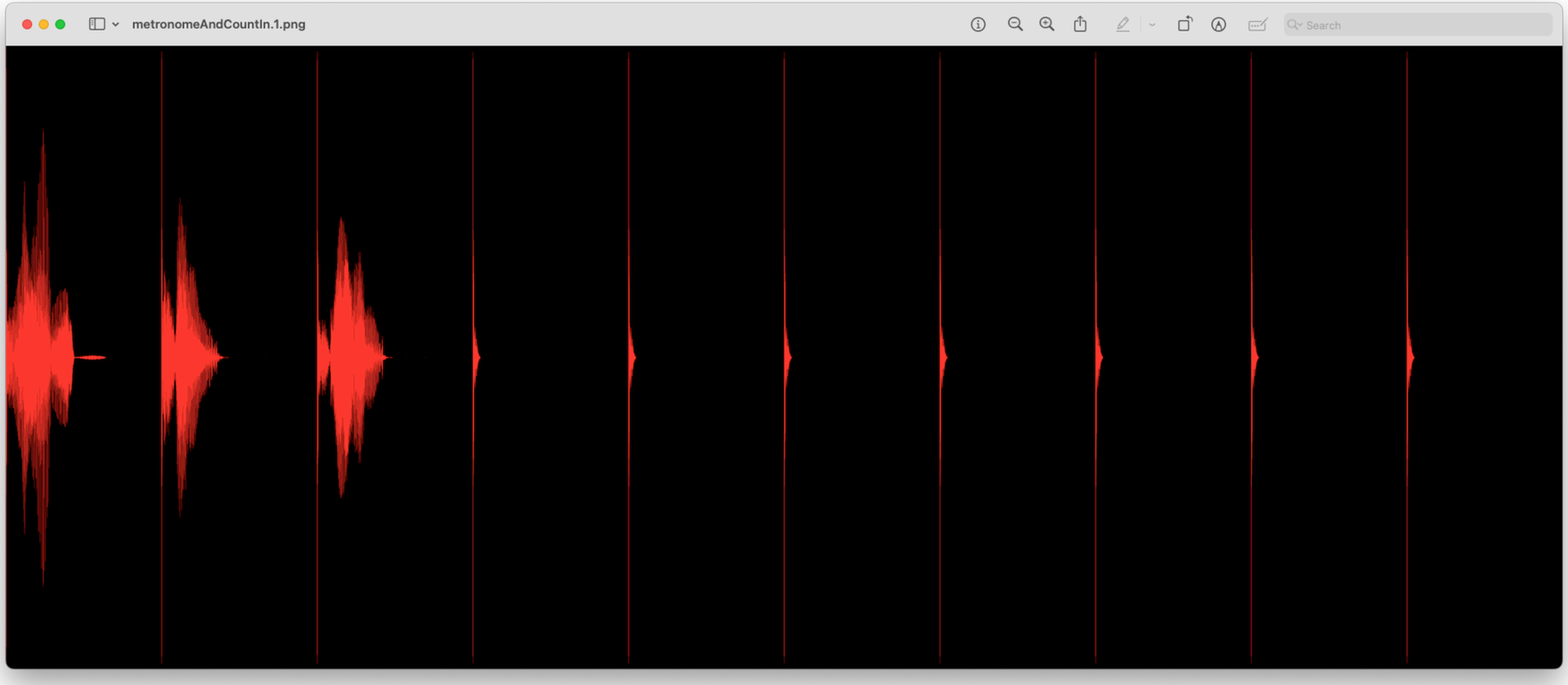
```swift
@Test(.snapshots(record: true))
func metronomeAndCountIn() async throws {
    let metro = Metronome()
    engine.connect(metro, to: engine.mainMixerNode, format: nil)

    let count = CountIn()
    engine.connect(count, to: engine.mainMixerNode, format: nil)

    try engine.render(bufferSize: bufferSize, format: format, to: buffer)

    assertSnapshot(of: buffer, as: .bufferImage(width: 10000, height: 4000))
}
```

```swift
@Test(.snapshots(record: true))
func metronomeAndCountIn() async throws {
    let metro = Metronome()
    engine.connect(metro, to: engine.mainMixerNode, format: nil)

    let count = CountIn()
    engine.connect(count, to: engine.mainMixerNode, format: nil)

    try engine.render(bufferSize: bufferSize, format: format, to: buffer)

    assertSnapshot(of: buffer, as: .bufferImage(width: 10000, height: 4000))
}
```

```swift
@Test(.snapshots(record: true))
func metronomeAndCountIn() async throws {
    let metro = Metronome()
    engine.connect(metro, to: engine.mainMixerNode, format: nil)

    let count = CountIn()
    engine.connect(count, to: engine.mainMixerNode, format: nil)

    try engine.render(bufferSize: bufferSize, format: format, to: buffer)

    assertSnapshot(of: buffer, as: .bufferImage(width: 10000, height: 4000))
}
```

```swift
@Test(.snapshots(record: true))
func metronomeAndCountIn() async throws {
    let metro = Metronome()
    engine.connect(metro, to: engine.mainMixerNode, format: nil)

    let count = CountIn()
    engine.connect(count, to: engine.mainMixerNode, format: nil)

    try engine.render(bufferSize: bufferSize, format: format, to: buffer)

    assertSnapshot(of: buffer, as: .bufferImage(width: 10000, height: 4000))
}
```


metronomeEverySecond.1.png
Search


metronomeAndCountIn.1.png
Search

Development process

- Don’t struggle with describing expected output
- Speed up the feedback cycle
- Finish with an artifact that serves as a test suite
- Nice Pull Requests

Try it out

Snapshot

Reference

Approval

Golden Master

Baseline

Characterisation

- https://github.com/jcavar/Metronome