Hier findest du die Prompts für Chat-GPT und den Code aus unserem Video zu Steinberg Cubase 14 ModScripter:
https://www.meinhomestudio.de/course/cubase-14-modskripter/

Prompt für Sinusmodulation:

Das folgende Script erzeugt ein Signal, mit dem sich beliebige Parameter in Audioeffekten modulieren lassen. Bitte schreibe Code, den ich dem hinzufügen kann, um eine Sinuswelle mit 1 Hz Geschwindigkeit auszugeben.
(Hier bestehenden Code aus dem ModScripter einfügen)

Code für Sinusmodulation:

```
// Globale Variable zur Phasenverfolgung
var phase = 0;

processModulation = function (inputValue, numSamples) {
    // Abtastrate abrufen
    var sampleRate = getSampleRate();

    // Phaseninkrement pro Sample berechnen
    var frequency = 1; // Frequenz in Hz
    var phaseIncrement = 2 * Math.PI * frequency / sampleRate;

    // Phase um numSamples vorrücken
    phase += phaseIncrement * numSamples;

    // Phase auf den Bereich [0, 2π] begrenzen
    phase %= 2 * Math.PI;

    // Sinuswert berechnen und auf [0,1] normalisieren
    var outputValue = 0.5 + 0.5 * Math.sin(phase);

    return outputValue;
}
```

Prompt für Pentatonik-Modulation:

Schreibe einen Code für ein neues Modul: Der Ausgang dieses Moduls ist unipolar und gibt im Zeitraum jeder 16tel-Note einen diskreten Wert aus. Hierbei werden die Werte von 0 bis 12 auf den Output gemappt, und es wird zu jeder 16tel-Note ein zufälliger Wert aus den folgenden verfügbaren Werten gewählt: 0, 2, 4, 7, 9, 12. Erzeuge selbst Zufallswerte im Moment jeder 16tel-Note und achte darauf, dass jede folgende Note einen anderen Wert annimmt.

Code für Pentatonik-Modulation:

```
// Globale Variablen zur Verfolgung der aktuellen 16tel-Note und des letzten Wertes
var lastSixteenthNoteIndex = -1;
var lastValue = null;

// Verfügbare Werte
var values = [0, 2, 4, 7, 9, 12];

processModulation = function (inputValue, numSamples) {
   // Aktuelle musikalische Position in Viertelnoten abrufen
   var projectTime = getProjectTimeMusic();

   // Dauer einer 16tel-Note in Viertelnoten
   var sixteenthNoteDuration = 0.25;

   // Aktuellen 16tel-Noten-Index berechnen
   var currentSixteenthNoteIndex = Math.floor(projectTime / sixteenthNoteDuration);

   // Prüfen, ob wir zu einer neuen 16tel-Note übergegangen sind
   if (currentSixteenthNoteIndex !== lastSixteenthNoteIndex) {
      // Mögliche Werte kopieren, um den letzten Wert ausschließen zu können
      var possibleValues = values.slice();

      // Letzten Wert entfernen, um Wiederholungen zu vermeiden
      if (lastValue !== null) {
         var index = possibleValues.indexOf(lastValue);
         if (index !== -1) {
            possibleValues.splice(index, 1);
         }
      }

      // Zufälligen Wert aus den verbleibenden Werten auswählen
      var randomIndex = Math.floor(Math.random() * possibleValues.length);
      var newValue = possibleValues[randomIndex];

      // Aktuellen Wert und 16tel-Noten-Index aktualisieren
      lastValue = newValue;
      lastSixteenthNoteIndex = currentSixteenthNoteIndex;
   }
```

```
    // Wert von 0-12 auf 0-1 normalisieren
    var outputValue = lastValue / 12;

    return outputValue;
}
```

Prompt für ADSR-Modulation:

Erstelle nun einen Code für ein völlig neues Modul, bei dem ein Output erzeugt wird, mit dem die Ausgangslautstärke eines Synths gesteuert wird, um 16tel-Noten synchron zum Projekttempo zu erzeugen, die zufällige unterschiedliche ADSR-Werte haben.
Der Ausgang ist unipolar und steuert den Gain-Parameter. Zu Beginn jeder 16tel-Note werden folgende Werte per Zufall festgelegt: Attack zwischen 0,1 ms und 20 ms, Decay zwischen 50 ms und 150 ms, Sustain zwischen 70% und 100% und Release zwischen 10 ms und 100 ms. Im Moment jedes Beginns einer 16tel-Note liegt der Output bei 0 und wird während der Attackzeit linear auf 1 erhöht. Direkt im Anschluss beginnt die Decay-Zeit und die Lautstärke wird innerhalb der Decay-Zeit von 1 auf den entsprechenden Sustainwert linear abgesenkt. Direkt nach Ablauf der Decay-Zeit beginnt die Release-Zeit. Der Output wird vom Sustain-Level innerhalb der Releasezeit linear auf 0 abgesenkt. Sollte in diesem gesamten Prozess bereits die nächste 16tel-Note beginnen, wird die aktuelle Modulation sofort abgebrochen und die nächste Attack-Zeit beginnt sofort und startet vom aktuellen Wert, anstatt von 0.

Cod für ADSR-Modulation:

```
// Globale Variablen
var lastSixteenthNoteIndex = -1;

var attackTime = 0;
var decayTime = 0;
var sustainLevel = 0;
var releaseTime = 0;

var envelopePhase = "idle"; // Mögliche Phasen: "attack", "decay", "release", "idle"
var phaseTime = 0; // Zeit seit Beginn der aktuellen Phase in Sekunden
var envelopeValue = 0; // Aktueller Wert der Hüllkurve

var previousEnvelopeValue = 0; // Startwert für die nächste Attack-Phase

// Hilfsfunktion für zufällige Werte zwischen min und max
function randomBetween(min, max) {
   return min + Math.random() * (max - min);
}

processModulation = function (inputValue, numSamples) {
   // Abtastrate abrufen
   var sampleRate = getSampleRate();
   var deltaTime = numSamples / sampleRate; // Zeitdauer seit dem letzten Aufruf in
Sekunden

   // Aktuelle musikalische Position in Viertelnoten
   var projectTime = getProjectTimeMusic();

   // Dauer einer 16tel-Note in Viertelnoten
   var sixteenthNoteDuration = 1 / 4; // 0.25
```

```
    // Aktuellen 16tel-Noten-Index berechnen
    var currentSixteenthNoteIndex = Math.floor(projectTime / sixteenthNoteDuration);

    // Prüfen, ob eine neue 16tel-Note begonnen hat
    if (currentSixteenthNoteIndex !== lastSixteenthNoteIndex) {
        // Vorherigen Hüllkurvenwert speichern
        previousEnvelopeValue = envelopeValue;

        // Zufällige ADSR-Werte generieren
        attackTime = randomBetween(0.1, 20) / 1000; // In Sekunden umrechnen
        decayTime = randomBetween(50, 150) / 1000;  // In Sekunden umrechnen
        sustainLevel = randomBetween(70, 100) / 100; // Prozent zu Dezimal
        releaseTime = randomBetween(10, 100) / 1000; // In Sekunden umrechnen

        // Hüllkurve neu starten
        envelopePhase = "attack";
        phaseTime = 0;

        // Aktuellen 16tel-Noten-Index aktualisieren
        lastSixteenthNoteIndex = currentSixteenthNoteIndex;
    }

    // Zeit seit der letzten Aktualisierung hinzufügen
    phaseTime += deltaTime;

    // Hüllkurvenberechnung basierend auf der aktuellen Phase
    switch (envelopePhase) {
        case "attack":
            var attackStartValue = previousEnvelopeValue;
            var attackEndValue = 1;

            if (attackTime > 0) {
                var attackProgress = phaseTime / attackTime;

                if (attackProgress >= 1) {
                    attackProgress = 1;
                    // Zur nächsten Phase wechseln
                    envelopePhase = "decay";
                    phaseTime -= attackTime;
                }

                envelopeValue = attackStartValue + (attackEndValue - attackStartValue) *
attackProgress;
            } else {
                // Sofortiger Attack
                envelopeValue = 1;
                envelopePhase = "decay";
```

```
            phaseTime = 0;
        }
        break;

    case "decay":
        if (decayTime > 0) {
            var decayProgress = phaseTime / decayTime;

            if (decayProgress >= 1) {
                decayProgress = 1;
                // Zur Release-Phase wechseln
                envelopePhase = "release";
                phaseTime -= decayTime;
            }

            envelopeValue = 1 + (sustainLevel - 1) * decayProgress;
        } else {
            // Sofortiger Decay
            envelopeValue = sustainLevel;
            envelopePhase = "release";
            phaseTime = 0;
        }
        break;

    case "release":
        if (releaseTime > 0) {
            var releaseProgress = phaseTime / releaseTime;

            if (releaseProgress >= 1) {
                releaseProgress = 1;
                // Hüllkurve beendet
                envelopePhase = "idle";
                envelopeValue = 0;
            } else {
                envelopeValue = sustainLevel * (1 - releaseProgress);
            }
        } else {
            // Sofortiger Release
            envelopeValue = 0;
            envelopePhase = "idle";
        }
        break;

    case "idle":
    default:
        envelopeValue = 0;
        break;
}
```

```
    // Sicherstellen, dass der Wert zwischen 0 und 1 liegt
    envelopeValue = Math.max(0, Math.min(1, envelopeValue));

    return envelopeValue;
}
```