Aunque tenemos que matizarlo aún, tendréis de plazo seguramente hasta el día 8 o 9 de mayo.

  \* Se os pedirá una pequeña memoria en la que detalléis: los integrantes del grupo, las contribuciones de cada uno, una breve descripción del proyecto realizado (de entre 100 y 300 palabras incluyendo detalles sobre la consecución de los objetivos propuestos, y extras no inicialmente planteados), tecnología utilizada (lenguajes, librerías, frameworks, plugins, etc.), y un par de ideas para la posible continuación/mejora del trabajo.

  \* Presentación o vídeo (8-10 minutos) + sesión de preguntas. Haremos dos o tres sesiones seguramente durante la semana del 8 de mayo.

Desarrollo de un Plugin de ritmos euclidianos con JUCE

Javier Cano Salcedo

Este proyecto nace de mi curiosidad por el desarrollo de extensiones de , con el objetivo final de que se pueda utilizar en una DAW, **Reaper** en mi caso.

Las características del plugin son las siguientes:

Desarrollo de un plugin VST en JUCE que lance eventos MIDI según patrones rítmicos euclidianos especificados mediante una división pulsos/compás, basado en HY-RPE2. Estos eventos podrán definirse directamente, aleatorizarse, relativizarse a otro canal o modificarse desde MIDI entrante. Al menos se podrán configurar 2 schedulers a la vez, pero me gustaría generalizarlo para n.

# Tecnologías utilizadas

## Juce

Juce es un framework de código abierto para desarrollar aplicaciones de audio para Windows, macOS, Linux, iOS y Android; y plugins VST, VST3, AU, AUv3, AAX y LV2; en C++.

## HY-RPE2

HY-RPE2 es un plugin secuenciador MIDI, con 2 modos de secuenciación: cuadriculada y euclidiana. A mí me interesa analizar la secuenciación euclidiana, que se presenta en esta ventana:

Interfaz de usuario gráfica

Descripción generada automáticamente

Lo que más llama la atención es la representación de ritmos euclidianos como una serie de circunferencias divididas concéntricas de colores. Cada circunferencia representa la fila de su color, y enviará un evento MIDI cuando la división que toque en ese momento esté coloreada. Voy a fijarme en los parámetros de cada fila y voy a intentar reproducirla.

# Desarrollo del plugin

Juce ofrece varias plantillas de proyectos. Para mi objetivo he seleccionado la plantilla “Plug-In>Basic”, que crea automáticamente archivos .h y .cpp de PluginProcessor y PluginEditor, que implementan en mi caso las clases JavierCano\_ProyectoFinalAudioProcessor y JavierCano\_ JavierCano\_ProyectoFinalAudioProcessorEditor, respectivamente. Por abreviar, las continuaré llamando PluginProcessor y PluginEditor.

La clase PluginProcessor es la que se encarga procesar los bloques de audio y MIDI. El plugin de ritmo euclidiano necesita generar eventos MIDI en los pulsos correspondientes. El problema se descompone en 2:

1. Calcular el reparto de pulsos según el algoritmo euclidiano
2. Enviar los eventos MIDI cuando correspondan

Gráfico de dispersión

Descripción generada automáticamente con confianza bajaGráfico

Descripción generada automáticamente con confianza mediaEl problema 1 se resuelve mediante una implementación del [**algoritmo de Toussaint**](https://en.wikipedia.org/wiki/Euclidean_rhythm#Summary_of_algorithm), que consiste en utilizar una matriz para transformar un vector de tamaño t comienza con todos los pulsos al principio y termina con el reparto más uniforme posible, donde t es el numero de tiempos y p es el numero de pulsos marcados en ese tiempo. Por ejemplo, para t = 13 y p = 5, inicializaríamos un vector con [x x x x x . . . . . . . .] y devolvería [x . . x . x . . x . x . .]. A nivel visual, se usa el algoritmo de Euclid para saber cuantas columnas de la matriz movemos debajo en cada paso, y finalmente se lee la matriz de arriba abajo.



vector<int> EuclideanRhythm::euclidAlgorithm(int steps, int pulses)

{

vector<int> euclidSequence;

euclidSequence.push\_back(pulses);

while (pulses > 0) {

//int q = steps / pulses;

int r = steps % pulses;

if(r != 0)

euclidSequence.push\_back(r);

steps = pulses;

pulses = r;

}

return euclidSequence;

}

vector<bool> EuclideanRhythm::toussaintAlgorithm(int steps, int pulses)

{

//Fuente: Toussaint's algorithm (https://en.wikipedia.org/wiki/Euclidean\_rhythm#Summary\_of\_algorithm)

int width = steps;

int height = 1;

//Empezamos con una matriz en la que todos los pulsos se dan primero y el resto se queda vacia

vector<vector<bool>> beatsDistribution(steps, vector<bool>(steps, false));

for (int i = 0; i < pulses; i++)

beatsDistribution[0][i] = true;

//Algoritmo de Euclid para minimizar el apelotonamiento

vector<int> euclidSequence = euclidAlgorithm(steps, pulses);

for (int i = 0; i < euclidSequence.size(); i++) {

//Cortamos y bajamos los ultimos tiempos de la matriz hasta que no se puedan bajar mas

for (int y = 0; y <= i; y++)

for (int x = 0; x < euclidSequence[i]; x++)

beatsDistribution[height + y][x] = beatsDistribution[y][width - 1 - x];

if (i + 1 < euclidSequence.size()) {

width -= euclidSequence[i];

height = steps / width;

if (steps % width != 0)

height++;

}

}

//Reconstruccion del rimto

vector<bool> rhythmBeats;

for (int x = 0; x < width; x++)

for (int y = 0; y < height; y++)

rhythmBeats.push\_back(beatsDistribution[y][x]);

rhythmBeats.resize(steps);

return rhythmBeats;

}

El problema 2

La clase PluginEditor es la que coloca y pinta los elementos gráficos en la ventana. Principalmente quiero que tenga una serie de filas como las de HY-RPE2, cada una representando un ritmo euclidiano. Para ello he creado una clase EuclideanRhythm que hereda de juce::Component.

#define ROWS\_NUMBER 4

EuclideanRhythm euclideanRhythm[ROWS\_NUMBER];

De esta manera, aunque el número de filas no sea una variable, programaré el plugin de manera generalizable para n filas. A continuación, muestro como queda con 4 filas u 8.

Interfaz de usuario gráfica

Descripción generada automáticamente

Interfaz de usuario gráfica

Descripción generada automáticamente

En estas capturas podemos ver todos los parámetros que tiene una clase EuclideanRhythm.

# Resultados

## Posibles mejoras