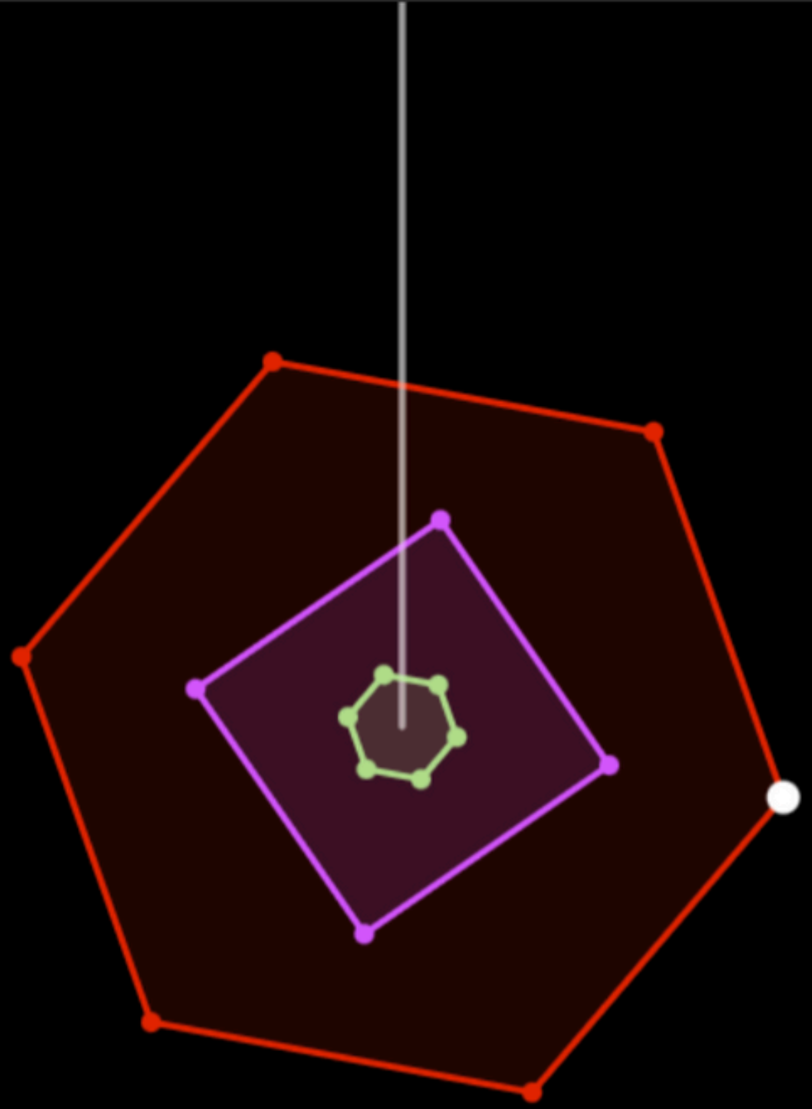
POLIMI - MAE

POLYDAW

POLYRHYTHM SEQUENCER

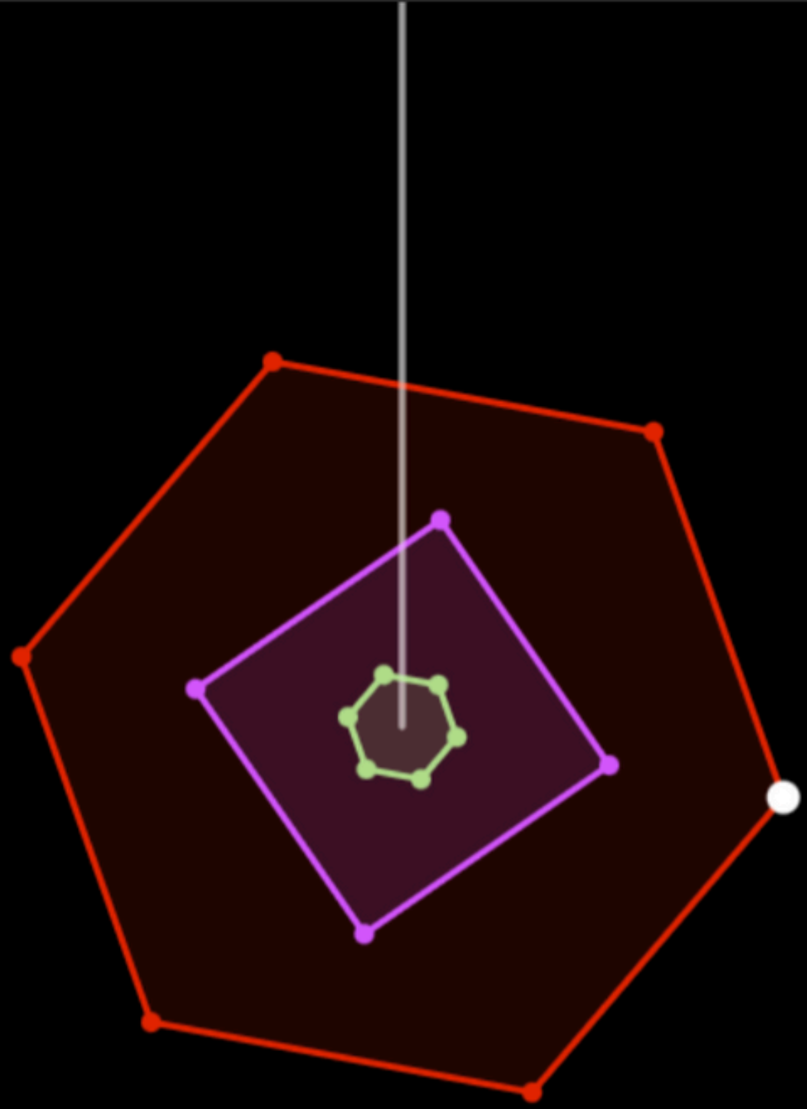
A web-based tool for visualizing and mastering polyrhythms.

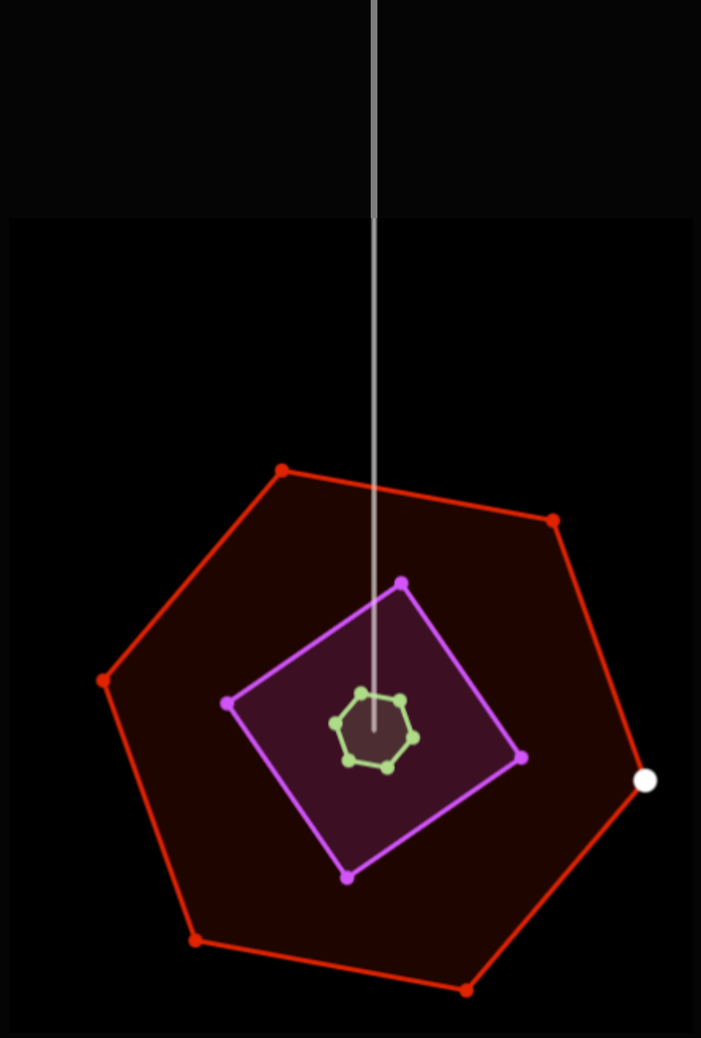
SANTIAGO COLORADO OCHOA
SAVERIO MARIANO BACINO
VICTOR-ALEXANDRU CORLAN
ANITHA SIVASANKAR



CORE CONCEPT

- Loops are represented by geometrical shapes, each corner being a step in the sequence.
- A triangle plays triplets; a square plays quarter notes...
- Allows for an easier approach to polyrhythms, just by plugging shapes in the interface.
- Oriented to make it easier to understand syncopation by watching shapes rotate rather than counting numbers.

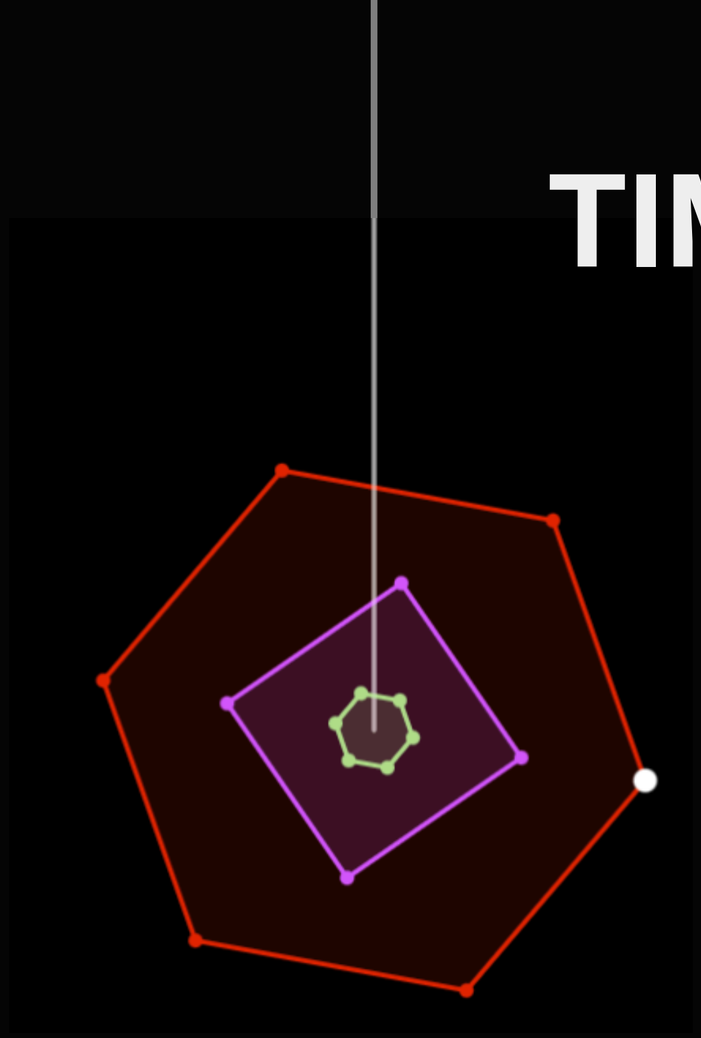




CLASS STRUCTURE: (RotatingPolygon)

- Each polygon is a self-contained instance managing its own geometry, physics state, and audio properties.
- Dynamic number of *sides*, *radius*, *volume*, and *corners*.
- *Radius* also controls the octave on which the polygon operates
- *Corners* is an array of objects which house information for the note it plays when it hits a certain corner, the volume it plays it at and the note's length as a fraction of available time

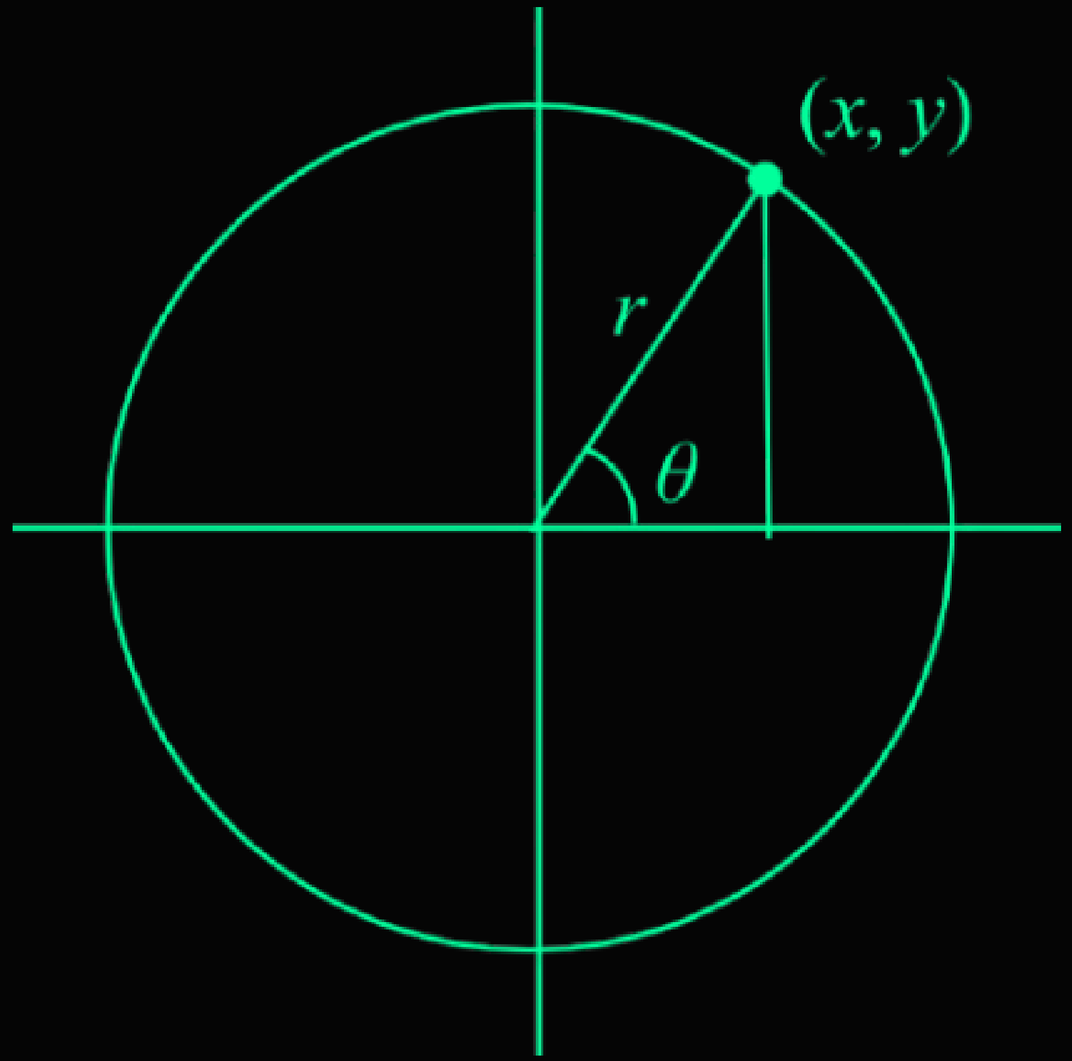
TIMING LOGIC & BPM SYNC

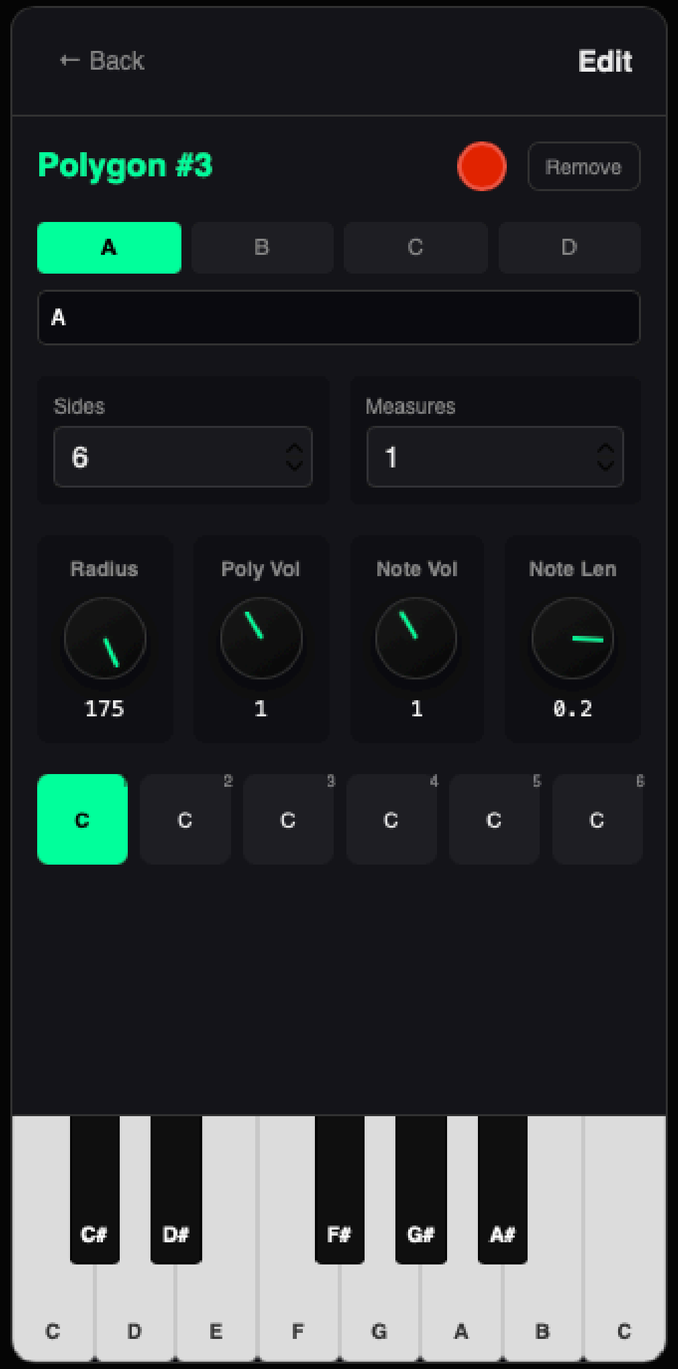


- Rotation is a function of *elapsedSeconds* rather than frame count, ensuring tempo accuracy regardless of screen refresh rate.
- All polygons reference the global *elapsedSeconds* variable, ensuring perfect phase alignment.
- The *Reset* button changes the *elapsedSeconds* variable to *-0.1* so that the first note can be detected
- Animation stops if the variable *isPlaying* is set to *false*, when the *Pause* button is pressed

INTERSECTION

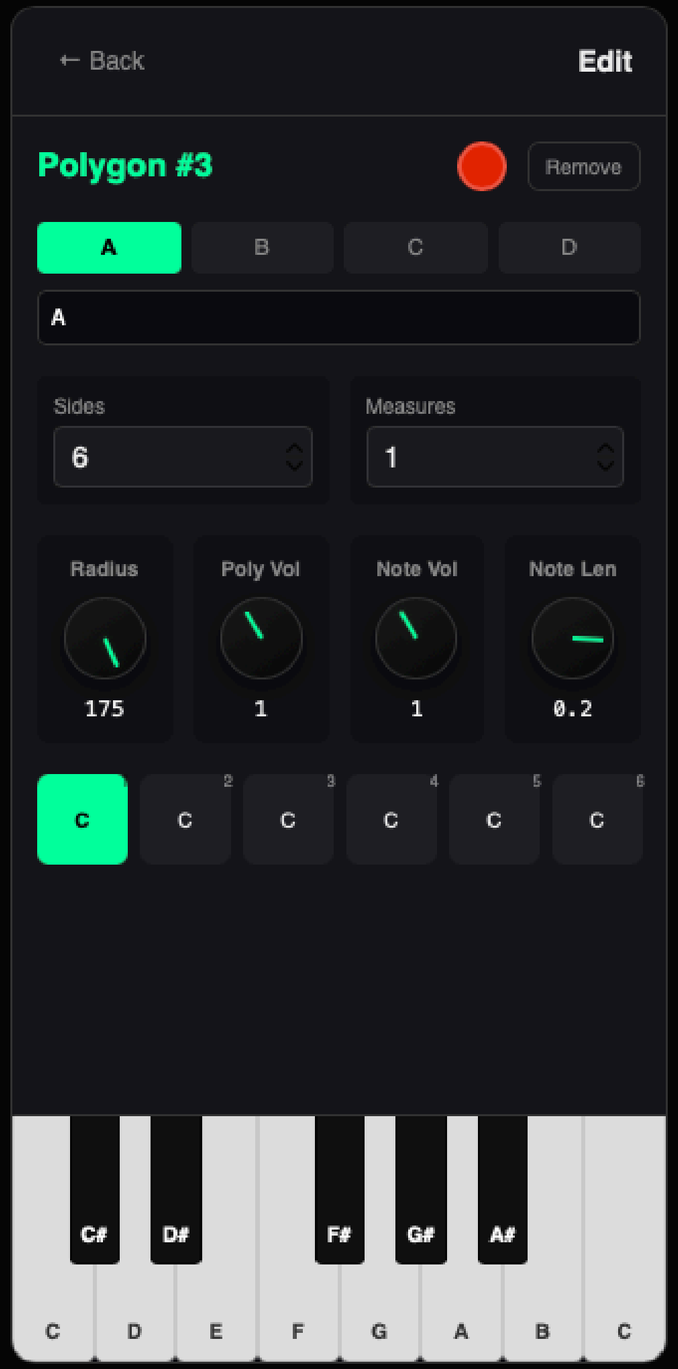
- To detect collision, we compare the angle of each corner to the top position (270 degrees going clockwise from the right)
- If a corner is within 4 degrees of the top of the polygon, its corresponding note gets played
- The flag ***wasHittingLine*** prevents the sound from triggering multiple times in a single frame.
- The middle line’s light-up animation starts within 12 degrees of the corner, and gets more intense the closer it gets to the center, depending on the number of sounds being played at the same time





SELECTING NOTES AND SETTING PARAMETERS

- Each polygon stores an indexed list of musical corners.
- The corner selector menu, rendered by *renderCornerDotsSelector()*, visually maps each vertex to a selectable dot.
- Clicking on a button selects the corner.
- The virtual keyboard outputs abstract notes, independent of the polygon structure.
- The function *assignNextCornerNote()* assigns the note to the selected corner and advances to the next corner in a cyclical way



PATTERN MANAGEMENT

- The app maintains a dictionary of patterns **(A, B, C, D)** within each polygon instance.
- State objects are cloned, allowing Pattern A to be modified without affecting Pattern B.
- Clicking a pattern button injects the saved state properties back into the active object instance.

MIDI GENERATION

- The code runs a numerical simulation of the physics loop
- It iterates through beats to calculate exact ***startTick*** timestamps for every collision event
- ***midi-writer-js*** converts these timestamps and note values into binary MIDI data (base64) for download
- The duration of a note is handled independently, as it can be changed through the options tab

```
  const MidiWriter = window.MidiWriter;

  const totalMeasures = lcmArray(polygons.map(p => p.measures));
  const totalBeats = totalMeasures * 4;

  const tracks = [];

  polygons.forEach((polygon, polyIndex) => {
    const track = new MidiWriter.Track();
    track.setTempo(globalBpm);

    const polygonDurationBeats = polygon.getRotationDuration();
    const repetitions = Math.ceil(totalBeats / polygonDurationBeats

    for (let r = 0; r < repetitions; r++) {
      polygon.corners.forEach(corner => {
        if (corner.note === 0) return;

        const midiNote = freqToMidi(getNoteByRadius(corner.note, po

        const deltaBeat = (corner.index / polygon.sides) * polygonD
polygonDurationBeats;

        const noteDurationTicks = polygon.getNoteDurationTicks() *

        track.addEvent(
          new MidiWriter.NoteEvent({
            pitch: [midiNote],
            duration: "T" + Math.round(noteDurationTicks),
            startTick: Math.round(deltaBeat * 480)
          })
        );
      });
    }
```

THANK YOU