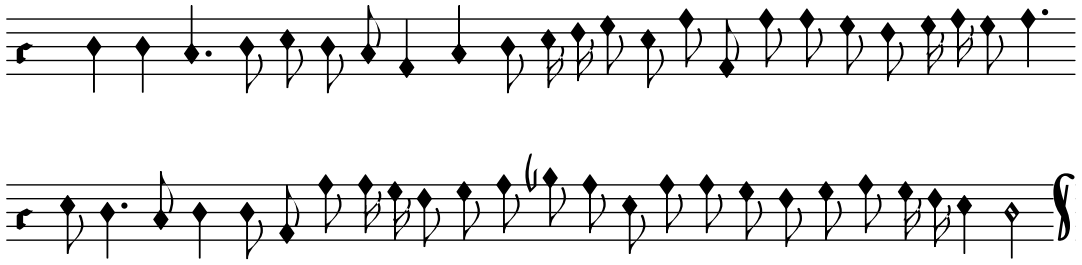
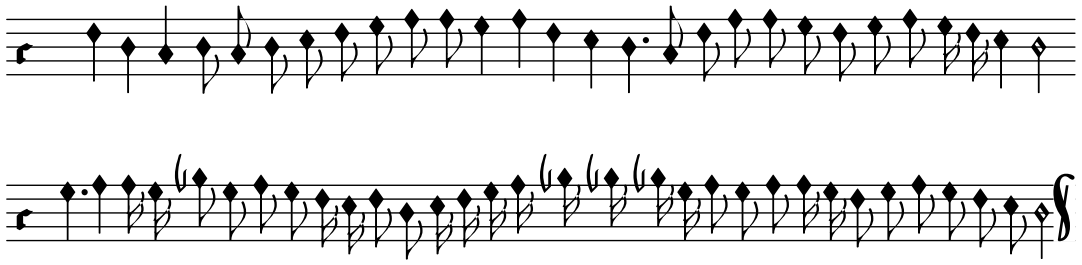
Sixteen Thoughts on Rule 110

Jeremy Mates

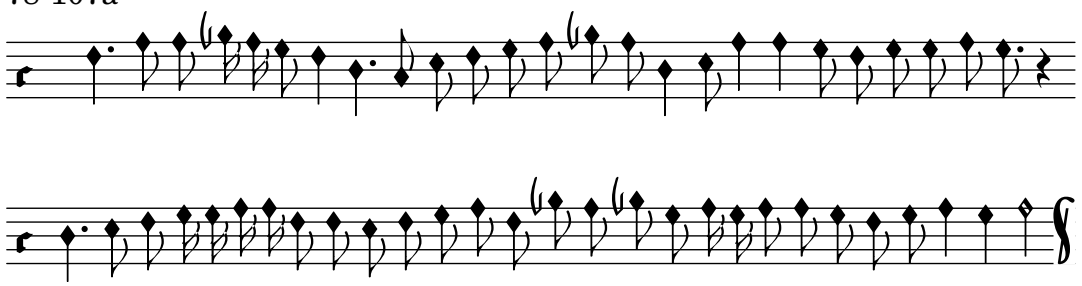
39-181a



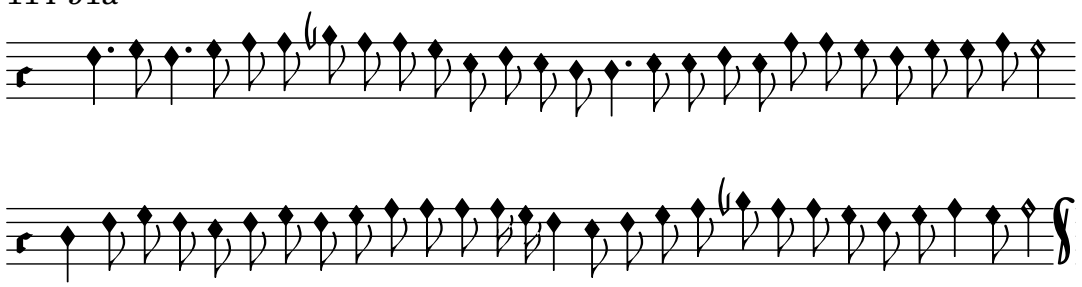
57-173a



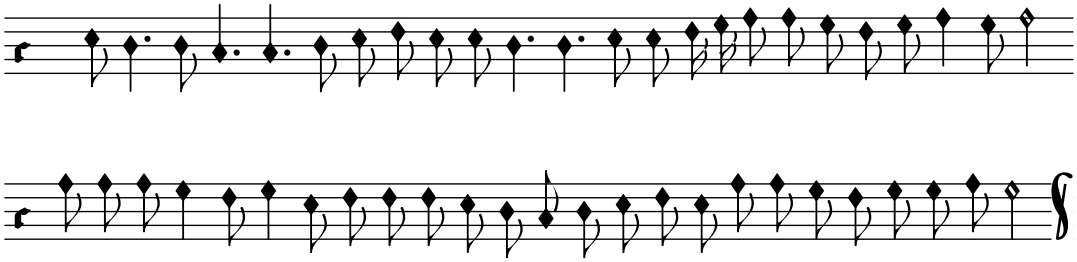
78-107a



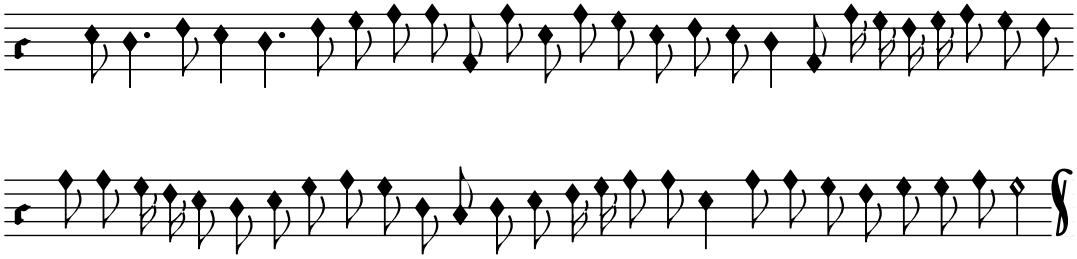
114-91a



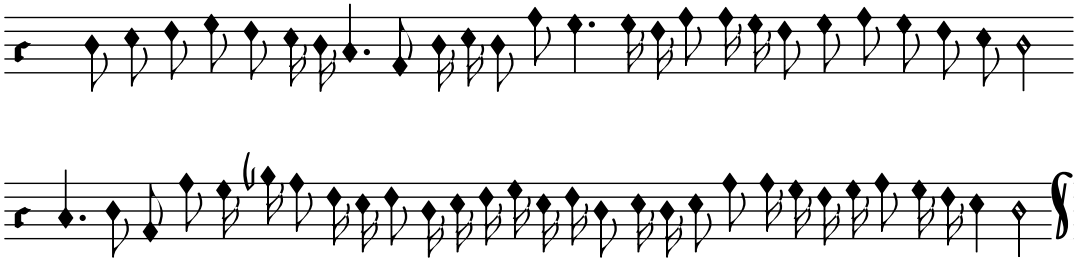
147-218a



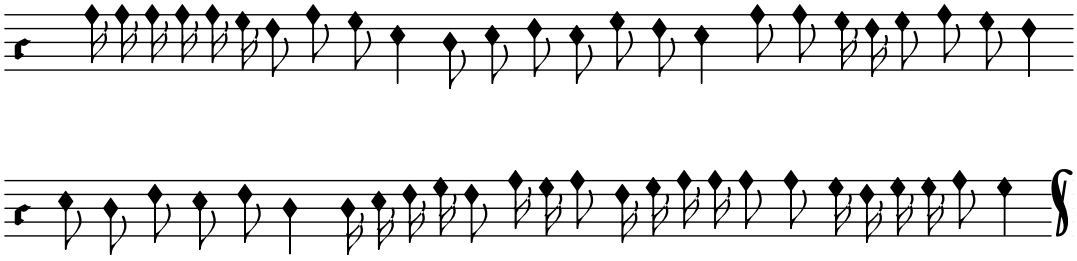
156-214a



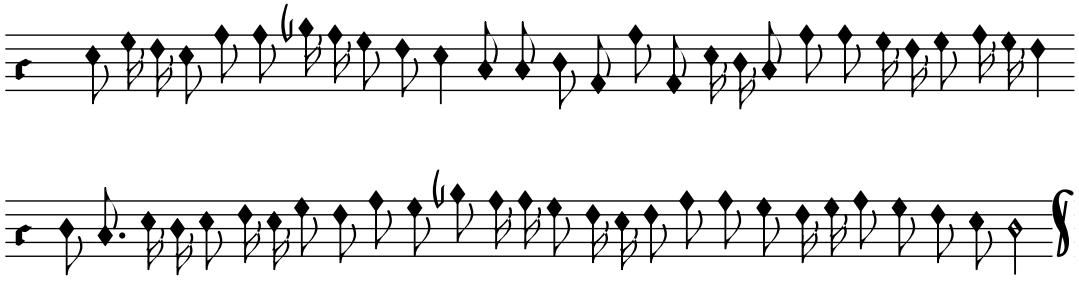
201-109a



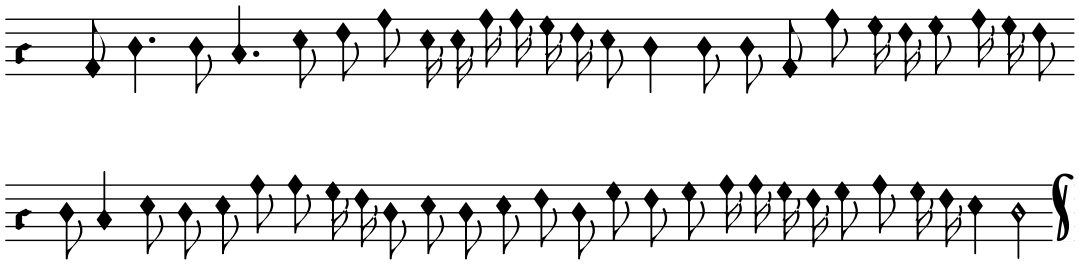
228-182a



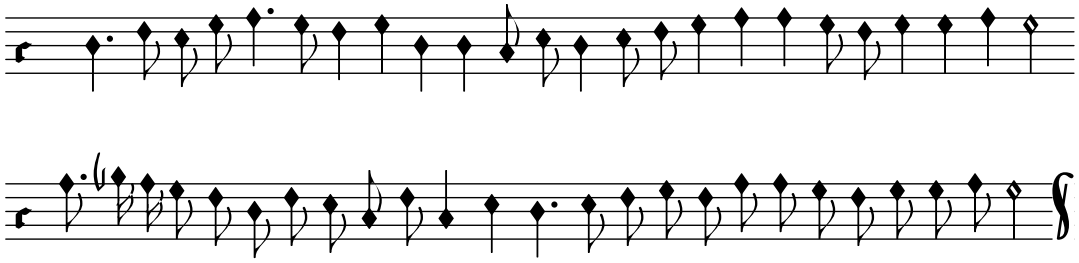
39-181b



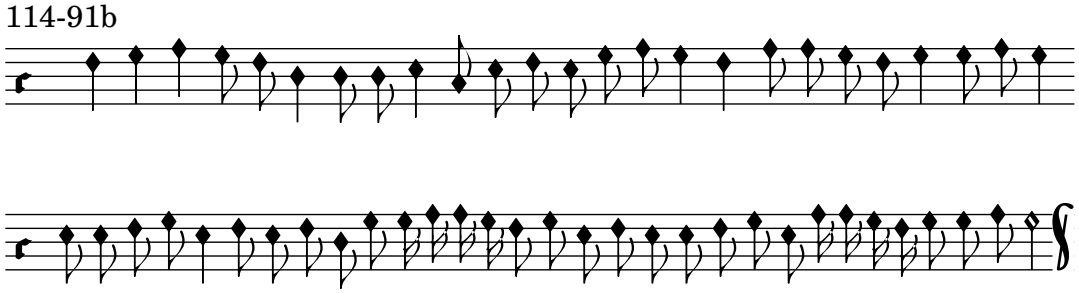
57-173b



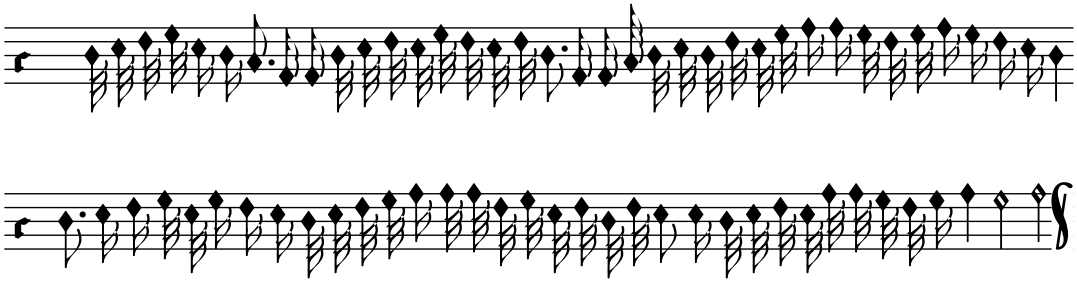
78-107b



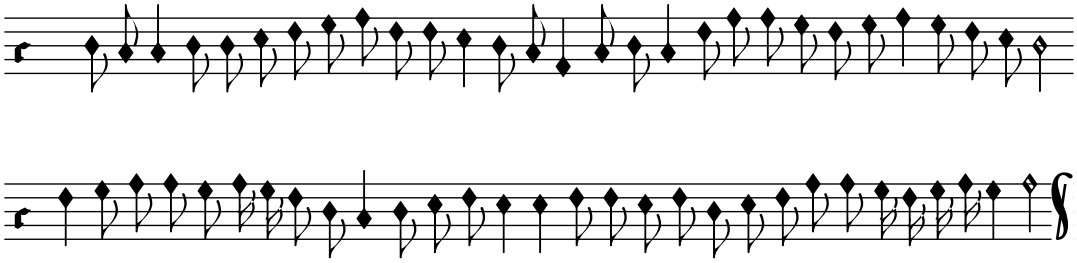
114-91b



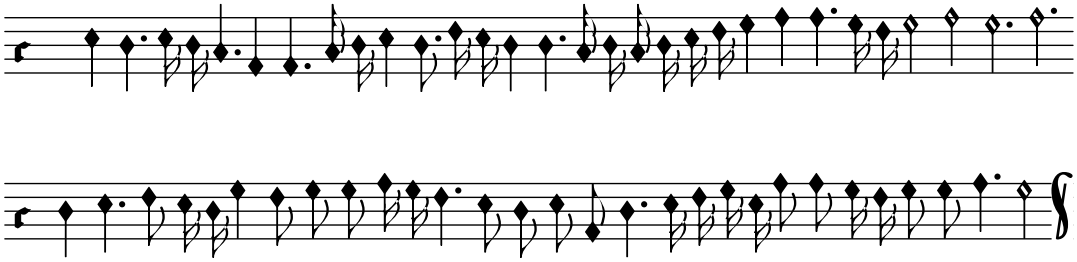
147-218b



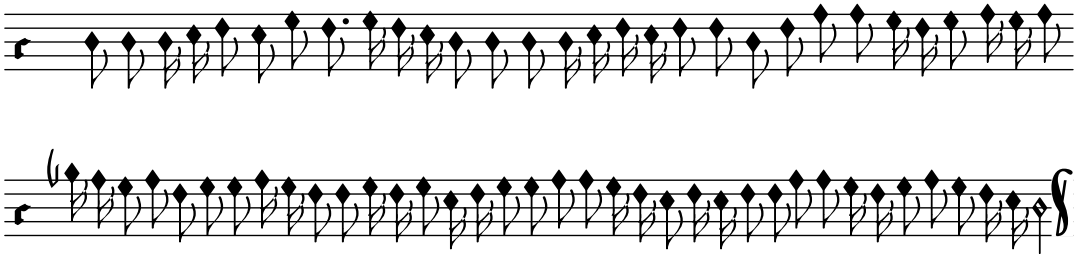
156-214b



201-109b



228-182b



Code

```
/*
 * Rule 110, 8-bit playfield.
 *
 * NOTE code written on a little-endian system and probably will need
 * porting to a system of some other endianess.
 *
 * Usage:
 *   cc -std=c99 -o gen gen.c && ./gen num
 *
 * where num is a number from 0 to 255 inclusive.
 */

#include <err.h>
#include <errno.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <sysexits.h>
#include <time.h>
#include <unistd.h>

#define FIELD_WIDTH 8
#define OUTCOMES_WIDTH 8

#define IFYOUHAVETOASK 42

int main(int argc, char *argv[])
{
    bool outcomes[OUTCOMES_WIDTH] = { 0, 1, 1, 1, 0, 1, 1, 0 };
    int index;
    uint8_t field1;
    unsigned int iters = 0;
    unsigned int input;

    if (argc != 2)
        errx(EX_USAGE, "Usage: gen {0..255}");

    argv++;                      // skip prog name
    if (sscanf(*argv++, "%3u", &input) != 1)
```

```
        errx(EX_USAGE, "Usage: gen {0..255}");
    argc -= 2;
    if (input > UINT8_MAX)
        errx(EX_DATAERR, "input number must be in range 0..%u", UINT8_MAX);
    field1 = input;

    while (++iters < IFYOUHAVETOASK) {
        uint8_t field2 = 0;
        for (unsigned int i = 0; i < FIELD_WIDTH; i++) {
            putchar(((field1 >> i) & 1) ? 'X' : '.');
        }
        printf(" %3u\n", (unsigned int) field1);

        for (int self = 0; self < FIELD_WIDTH; self++) {
            int prev = self - 1;
            int next = self + 1;
            /* Wrap the field around so do not have to worry about
             * (undefined) handling of edge conditions. */
            if (prev < 0)
                prev = FIELD_WIDTH - 1;
            if (next > FIELD_WIDTH - 1)
                next = 0;

            index =
                ((field1 >> prev & 1) << 2) | ((field1 >> self & 1) << 1) |
                (field1 >> next & 1);
            field2 |= outcomes[index] << self;
        }
        field1 = field2;

        if (field1 == 0 || field1 == UINT8_MAX) {
            printf("hilo %u at %u\n", field1, iters);
            exit(0);
        }
    }
    printf("loop %u\n", iters);
    exit(0);
}
```

Method

The use of an 8-bit ring buffer keeps output minimal (32-bit has a much larger space to explore), and allows easy graphing of the resulting patterns. The melodies above are based on the eight starting numbers that have a single subsequent step before terminating. There are other interesting patterns in some of the remaining numbers perhaps worth exploring.

```
$ ./gen 39
XXX..X..  39
X.X.XX.X 181
hilo 255 at 2
```

Melodization was performed by taking the numbers as binary:

```
11100100 39
10101101 181
```

Then spacing these numbers out:

```
 1   1   1   0   0   1   0   0
 1   0   1   0   1   1   0   1
```

A new number created with the previous, current, and next numbers for each number, plus `s` and `e` to indicate the start and end:

```
 1   1   1   0   0   1   0   0
s11 111 110 100 001 010 100 00e
 1   0   1   0   1   1   0   1
s10 101 010 101 011 110 101 01e
```

And then each set (`s11`, `111`, etc.) is considered in the construction of a musical phrase. Particular codas[Taruskin, 2010, p.29] were used for the four `e` cases, though otherwise the process was fairly loose, with a `1` usually meaning ascending or faster, and `0` the opposite, and a variable number of notes or overlap allowed between the different sets. So, a fairly informal process. Whether a cellular automata melody[Stephenson, 2008, p.600] could be designed that is both musically pleasing and possible to recover the original bits from might make for an interesting study.

Bibliography

Neal Stephenson. *Anathem*. HarperCollins, 2008. ISBN 978-0-06-147409-5.

Richard Taruskin. *Music from the Earliest Notation to the Sixteenth Century*. Oxford University Press, 2010. ISBN 978-0-19-538481-9.