Desenvolupament d'un *cellular automaton* controlat per Raspberry Pi

18 d'octubre del 2018

Joan Pallarès Albanell

Índex

0. Resum....3
1. Prefaci....3
2. *Game of Life: Cellular automata* en 2D....3
3. *Cellular automata* en 1D....4
4. Objectiu....4
5. Hardware utilitzat....5
6. Software utilitzat o desenvolupat....6
    a. Instal·lar el driver MAX7219....6
    b. Descomposar la llibreria....6
    c. Desenvolupar un *cellular automaton* 1D seguint la *rule90*....6
7. Discussió....8
8. Bibliografia/Webgrafia....10

0. Resum

Els *cellular automata* són un tipus de jocs sense cap jugador que ens permeten observar l'evolució de formes de vida fictícies seguint unes regles preestablertes. El projecte que presentem en aquesta memòria consisteix en el desenvolupament d'un *cellular automaton* en una matriu LED de 8x8 connectada a una Raspberry Pi. L'obediència a les regles que determinen l'evolució dels organismes artificials es determina mitjançanant un script desenvolupat en python3.

1. Prefaci

La idea d'aquest projecte la vaig extreure d'una revista de Raspberry Pi. Allà hi presentaven un projecte d'un informàtic anomenat Ferran Fàbregas: la *LifeBox*. Aquell projecte consistia en una matriu de 32x32 LEDs RGB controlada per una Raspberry. Mitjançant un script en C, el projecte intentava simular les evolucions de diverses espècies vives representades pels diferents colors.

Aquest projecte em va fer recordar el *Game of Life*. Un *joc* que, pel que tenia entès, recreava vida microscòpica per ordinador. A l'article sobre el *LifeBox* deien que aquest projecte no tenia res a veure amb el *Game of Life*. Però les similituds són òbvies, i a mi em van inspirar el projecte que presentem en aquesta memòria. La idea inicial era utilitzar un muntatge electrònic inspirat en el del Ferran Fàbregas i en els plantejaments del *Game of Life*.

Aviat, però, vaig optar per establir com a objectiu una versió simplificada d'aquesta idea inicial. Per començar, la matriu era molt més senzilla, només tenia un color i era de 8x8. A més, vam optar per una variant més senzilla del *Game of Life*, que seguia la mateixa lògica però en 1D enlloc de en 2D. Sigui com sigui, aquest projecte ha servit per endinsar-nos en les possibilitats de la Raspberry Pi i del món entorn al Game of Life.

2. *Game of Life: Cellular automata* en 2D

El *Game of Life* és un univers fictici dissenyat pel matemàtic britànic John Horton Conway. En aquest univers, unes cèl·lules habiten una quadrícula 2D infinita, on es reprodueixen i moren seguint unes lleis preestablertes. El *Game of Life* és un joc, però es tracta d'un joc sense cap jugador (*zero-player game*). Segons la Wikipedia, aquest tipus de jocs es caracteritzen perquè no hi ha cap jugador sensible (*senitent*). El *Game of Life* pertany a un subgrup de jocs sense cap jugador que s'anomena *cellular automaton*: sistemes dinàmics, deterministes i discrets. Els *cellular automata*, més enllà de la seva utilitat immediata com a jocs, poden ser utilitzats per a modelar sistemes naturals (Kari, 1990). Però només si considerem que aquests estan formats per una gran quantitat d'objectes simples que interactuen localment.

Martin Gardner va popularitzar el *Game of Life* en un article de l'any 1970 a la revista Scientific American. Segons Gardner, el nom del joc era degut a que consistia en una analogia dels canvis poblacionals que tenen lloc en una societat d'organismes vius. De fet, l'objectiu del joc no és altre que obsevar l'evolució de les figures que formen els conjunts d'organismes unicel·lulars al llarg del

temps. En el seu article, Gardner descrivia les regles que Conway havia establert per determinar els naixements i morts dels organismes que habitaven el seu univers. Aquestes regles s'apliquen discretament, per generacions. Conway va esforçar-se en que aquestes regles fessin que el comportament de la població fos *inpredictible*.

Aquestes regles relacionen cada quadrat amb els vuit quadrats adjactents, i determinen si la cèl·lula que el conté estarà viva o morta en la següent generació. Les regles són les següents:

1. Supervivències. Cada quadrat amb dues o tres cèl·lules veïnes vives sobreviu, passa a la següent generació.
2. Morts. Cada quadrat amb quatre o més cèl·lules veïnes vives mor degut a la sobrepoblació. Cada quadrat amb una cèl·lula veïna viva o menys mor d'aïllament.
3. Naixements. Hi ha un naixement en cada quadrat buit adjacent a tres cèl·lules vives veïnes.

Quan es va publicar aquest article l'any 1970, només l'estudi d'algunes figures complexes que apareixien al llarg de les generacions començava a simular-se per ordinador. El més habitual era jugar al *Game of Life* amb paper i bolígraf, o utilitzant el taulell d'algun joc de taula i fitxes de colors. Això feia que el procés fos lent, tediós i que els errors fossin força freqüents. Avui en dia, amb el desenvolupament i la ubiqüitat de la informàtica, les simulacions del *Game of Life* són molt més ràpides, àgils i precises.

3. *Cellular automata* en 1D

Segurament el *Game of Life* és el *cellular automaton* més famós, però n'hi ha d'altres. Així, en un treball publicat l'any 1984, Martin, Odlyzko i Wolfram analitzaven diversos tipus de *cellular automata*. Entre ells, hi analitzaven diferents variants de *cellular automata* en 1D. En aquest cas, la viavilitat de cada cèl·lula es determina per l'estatus (viva o morta) de les cèl·lules en les dues quadrícules adjacents. Tot i la naturalesa 1D d'aquesta aproximació, l'evolució al llarg del temps se sol representar en 2D. Això s'aconsegueix utilitzant una quadrícula, on cada fila és una generació, amb la primera generació en la part superior de la quadrícula.

En aquest treball ens centrarem en una tipus específic de *cellular automaton* en 1D, el conegut com a *rule90.* Aquesta regla està basada en l'operació lògica XOR: l'output és veritat només si els dos inputs són diferents. La *rule90* porta aquest nom perquè l'output del conjunt dels inputs possibles es pot resumir en 01011010, que és 90 en representació binària (Figura 1).

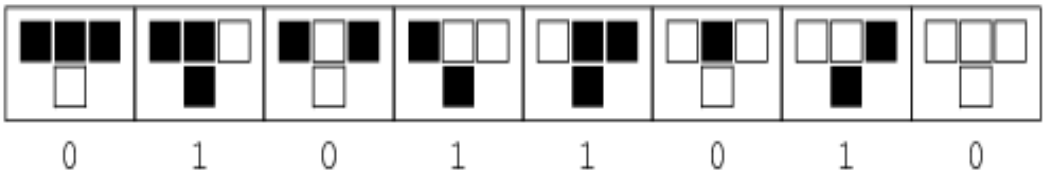

Figura 1. *Rule90*. La fila superior representa totes les combinacions possibles d'una cèl·lula central (viva o morta) i les seves dues veïnes (vives o mortes). La fila inferior representa l'output, seguint la lògica XOR, de cada una de les situacions de la fila superior. Imatge adaptada de http://mathworld.wolfram.com/Rule90.html

4. Objectiu

L'objectiu del projecte que presentem és desenvolupar, en python3, un *cellular automaton* que segueixi la *rule90*. Les evolucions d'aquest cellular automaton volem que es mostrin en una matriu LED 8x8 connectada a una Raspberry Pi 3. En concret, volem que la primera generació es mostri en la fila superior i en cada fila següent s'hi mostri el canvi poblacional d'aquella generació.

5. Hardware utilitzat

La representació gràfica del *cellular automaton* en 1D l'hem fet en una matriu LED VMP312 amb un xip MAX7219. O sigui, que enlloc d'una quadrícula infinita, tenim una matriu de 8x8 LEDs. Aquesta matriu l'hem connectat a un ordinador Raspberry Pi3 Model B+. El conjunt s'ha connectat mitjançant una bread board i es mostra en la Figura 2.

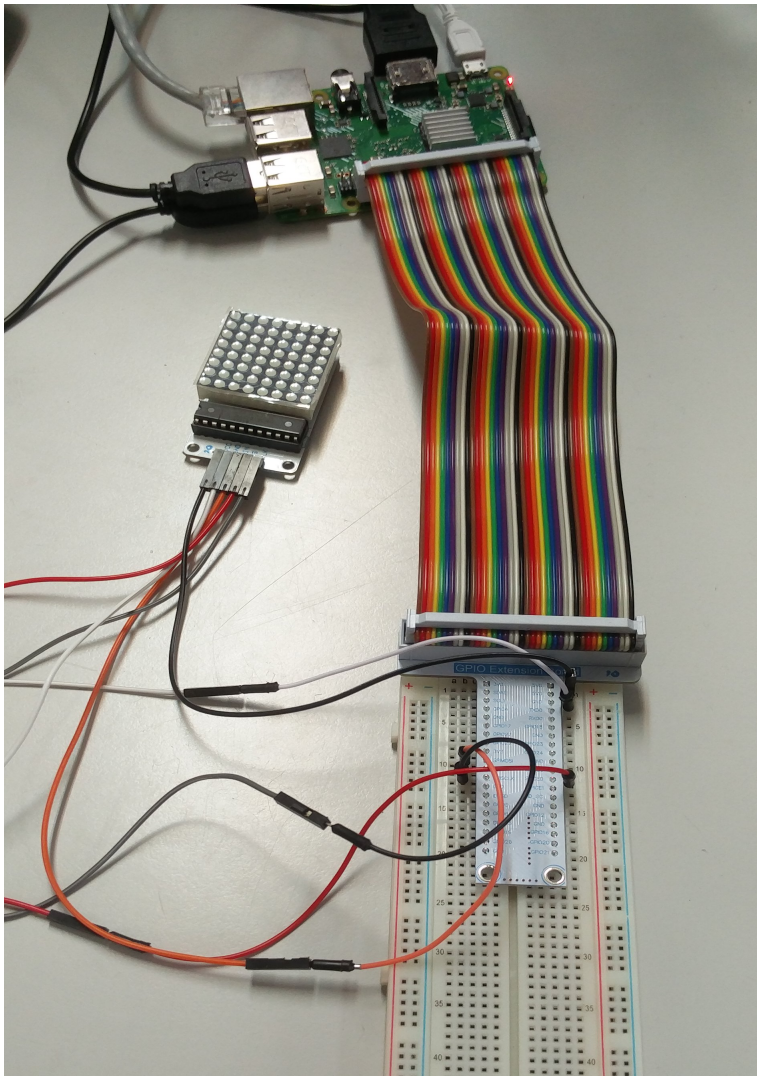

Figura 2. Muntatge electrònic consistent en una Raspberry Pi3 Model B+, una breadboard i una matriu LED VMP312 amb un xip MAX7219.

6. Software utilitzat o desenvolupat

a. Instal·lar el driver MAX7219

El primer pas va ser instal·lar un driver per al xip MAX7219, això ho vam fer amb les comandes següents:

```
$ git clone https://github.com/rm-hull/max7219.git
```

Amb aquesta comanda obtenim una carpeta anomenada max219 on hi ha el driver, a més d'altres elements interessants com programes d'exemple per a utilitzar la matriu de LEDs.

A continuació ens movem a la carpeta max7219:

```
$ cd max7219/
```

I executem la comanda següent per a instal·lar el driver:

```
$ sudo python3 setup.py install
```

b. Descomposar la llibreria

Amb el driver instal·lat, vam ser capaços d'utilitzar l'script d'exemple matrix_demo.py, que venia amb el driver, per a mostrar missatges predeterminats en la matriu. Modificant mínimament l'script de matrix_demo.py vam poder mostrar qualsevol missatge en la matriu. Però per a implementar un *cellular automaton* això no era suficient. Necessitavem tenir control a nivell individual de cada un dels LEDs que formen la matriu. Ja que cada LED seria el quadrat on habitaria cada una de les cèl·lues. Descomposant els elements que formen la llibreria *luma* de python, i gràcies a l'ajuda del docent, vam poder controlar l'encesa o apagada de LEDs específics. Les línies bàsiques de python3 per a fer això són:

```
with canvas(virtual) as draw:
    draw.point(punt,fill=1)
```

Aquí, *punt* són les coordenades del LED específic que volem encendre, *fill=1* determina que el led s'encendrà, *fill=0* faria que no s'encengués.

c. Desenvolupar un *cellular automaton* 1D seguint la *rule90*

Centrant-nos en aquestes línies bàsiques, vam desenvolupar un script en python3 anomenat rule90_matrix.py. Aquest script mostra un *cellular automaton* 1D seguint la *rule90* en la matriu de LEDs. En la fila superior s'encenen uns LEDs a l'atzar, les progressives generacions d'aquests LEDs, d'aquestes cèl·lules, es mostren a cada fila. Aquest n'és el codi:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright (c) 2017-18 Richard Hull and contributors
# See LICENSE.rst for details.
#My modifications from the teacher-modified script

#Obscure
import keyboard
from luma.led_matrix.device import max7219
from luma.core.interface.serial import spi, noop
from luma.core.render import canvas
from luma.core.virtual import viewport

#Understandable
import time as t
import random as r

serial = spi(port=0, device=0, gpio=noop())
device = max7219(serial, cascaded=1 or 1, block_orientation=0, rotate= 0)
virtual = viewport(device, width=10, height=10)

#I can start playing

##Set row 0, the INITIAL STATE row. The first row, is randomly generated.

cs=[] #current state list
pd={} #provisional dictionary to store the cell state at the specified
position
bd={} #BIG DICTIONARY to store EVERYTHING so the LEDs are not turned off
at each row change. In fact they are switched off, but they are switched
on again, as previous rows are stored in bd
with canvas(virtual) as draw:
    for column in range(8):
         cell_state=r.randrange(2) #Randomly determine if cells are alive
(1) or dead (0)
           draw.point((column,0), fill=cell_state) #for each column from
row=0
        bd[column, 0]=cell_state #BIG DICTIONARY
        print("This is bd\n",bd,"\nIts length is",len(bd),"\n")
        pd[column]=cell_state
        cs.append(pd)
        pd={} #reset pd
t.sleep(2)
print("This is the current state (cs)",cs,"\n")

###Develop the INITIAL STATE on the subsequent rows, following XOR (the
output is True only if the two inputs are different)

for row in range(1,8): #row 0 is were the INITIAL STATE is placed, so we
must start at row=1 here
    dcs=[] #developing current state list
    with canvas(virtual) as draw:
        for column in range(8):
            if column==0 or column==7: #Columns on the extremes only have
one neighbour, so they require special rules. I decided they should
change they status at every row
```

```
                if cs[column][column]==0:
                    cell_state=1
                else:
                    cell_state=0
            else:
                 #print("This is row",row,"This is column",column,", this
is cs[column-1][column-1]",cs[column-1][column-1],", this is cs[column+1]
[column+1]",cs[column+1][column+1])
                  if cs[column-1][column-1]!=cs[column+1][column+1]: #If
the two neighbours of the current cell are different, this cell will be
alive in the next generation.
                    cell_state=1
                else:
                    cell_state=0
            bd[column, row]=cell_state #BIG DICTIONARY
            pd[column]=cell_state
            dcs.append(pd)
            pd={} #reset pd
    cs=dcs #the developing current state becomes the current state
    print("This is bd\n",bd,"\nIts length is",len(bd),"\n")
    with canvas(virtual) as draw:
         for turn_on_row in range(row+1): #THE BIG DICTIONARY IN ACTION.
+1 is to include the row just developed above
            for turn_on_column in range(8):
                              draw.point((turn_on_column,  turn_on_row),
fill=bd[turn_on_column,  turn_on_row]) #for  each  position  use  the  BIG
DICTIONRY to know if it should be on or off
                print("turn_on_column=",turn_on_column,"turn_on_row=",tur
n_on_row,"bd[turn_on_column,             turn_on_row]",bd[turn_on_column,
turn_on_row])
    t.sleep(2)

t.sleep(10)
```

El problema més gran a l'hora de desenvolupar el codi va ser aconseguir que, en passar de fila, els LEDs de les altres no s'apaguessin. Això es va aconseguir amb un diccionari (a l'script anomenat *bd*, *big dictionary*). Aquest diccionari emmagatzema la informació per a cada generació un cop es determina quines cèl·lues estan vives i quines no. La informació continguda en aquest diccionari serveix per a encendre tots els LEDs que s'han encès en la generació actual i en totes les precedents. Això crea l'efecte que els LEDs no s'apaguen, però de fet s'apaguen i es tornen a encendre a cada generació. Un problema menor va ser determinar què passa amb les cèl·lules als extrems de la matriu. Els *cellular automata* estan pensats per a operar en una quadrícula infinita, aquí ens veiem limitats per una matriu de 8x8. Vam obtar per establir unes regles especials per a les cèl·lules situades als extrems: canviarien d'status a cada generació. O sigui, si una cèl·lula situada a l'extrem de la matriu està viva a la següent generació passa a estar morta, i a la inversa.

7. Discussió

La primera conclusió és que l'script funciona. Tal i com mostra la Figura 3, a la fila superior s'hi encenen aleatòriament alguns LEDs que representen una població inicial de cèl·lules. Seguint la *rule90*, aquesta població inicial va desenvolupant-se, generació a generació, en les files següents. Per tant, s'ha assolit l'objectiu d'implementar un *cellular automaton* 1D seguint la *rule90*.

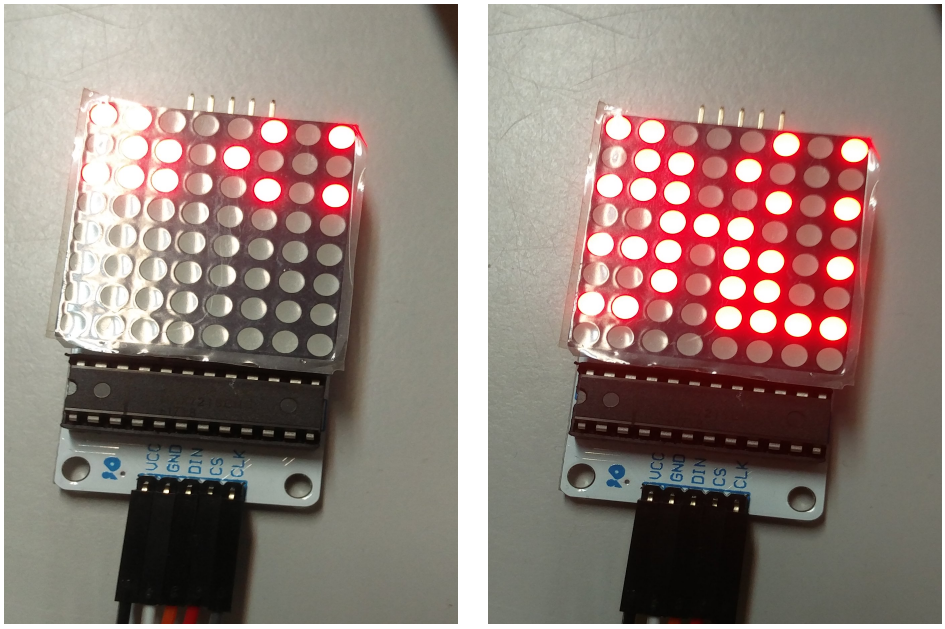

Figura 3. Imatges de l'script rule90_matrix.py en funcionament. La fotografia de l'esquerra i la de la dreta mostren dos estadis diferents del mateix procés. De dalt abaix, cada fila representa una generació. Dins d'una fila, cada LED encès representa una cèl·lula viva en aquella generació. Cada generació depèn de l'anterior seguint la *rule90*. La primera generació, en la fila superior, es generada a l'atzar.

Més enllà d'aixo, es fa evident ben aviat que una matriu 8x8 és un suport molt limitat per a implementar un *cellular automaton* que està concebut en una quadrícula infinta. Una millora molt simple d'aquest projecte a nivell de Hardware podria ser utilitzar una matriu més gran. Les modificacions que caldria fer a l'script per adapatar-lo a aquesta matriu serien mínimes.

A nivell de software, tot i la satisfacció d'haver assolit l'objectiu proposat, la manca de temps ens ha impossibilitat investigar a fons els *cellular automata*. Sí que voldríem, però, proposar algunes millores i alternatives al projecte descrit aquí. La primera i més senzilla seria fer un script d'arrancada en bash i guardar-lo a la carepeta /etc/init.d per a que el nostre script arrenqui tan bon punt s'encengui la Raspberry. Això comportaria que el nostre *cellular automaton* podria funcionar sense necessitat de pantalla, ratolí o teclat. Una modificació de l'script rule90_matrix.py per a que tot el procés s'incerís dins d'un *while* infinit podria complementar l'script d'arrancada i dotaria de més autonomia el *cellular automaton*.

D'altra banda, es podria modificar l'script rule90_matrix.py per a que seguís una altra lògica. Per exemple, enlloc de seguir la *rule90* podria seguir la *rule30*. Anàlogament a la *rule90*, la *rule30* s'anomena així perquè l'output del conjunt dels inputs possibles es pot resumir en 00011110, que és 30 en representació binària.

Per últim, en el moment de finalitzar aquesta memòria, hem començat a treballar en una versió del Game of Life clàssic. L'script està en desenvolupament i es pot veure en l'arxiu adjunt anomenat game_of_life.py. En definitiva, considerem que el projecte ha assolit l'objectiu plantejat. Però, sobretot, ens ha permès explorar les possibillitats de la Raspberry i acostar-nos al món dels *cellular automata*.

8. Bibliografia/Webgrafia

1. Zero-player game. Wikipedia, https://en.wikipedia.org/wiki/Zero-player_game
2. J. Kari, 1990. Reversibility of 2D cellular automata is undecidable. Physica D: Nonlinear Phenomena
3. M. Gardner, 1970. The fantastic combinations of John Conway's new solitaire game "life". Scinetific American
4. O. Martin, A. Odlyzk and S. Wolfram, 1984. Algebraic Properties of Cellular Automata. Communications in Mathematical Physics http://mathworld.wolfram.com/Rule90.html