Κουτρας Δημήτριος
ΑΕΜ 355

Για λόγους ευκολίας μπορούμε να περάσουμε μέσω της γραμμής εντολών διαφορετικίες παραμέτρους στο πρόβλημα μας. Γράφωντας την εντολη *"python runner.py –help" εμφανίζεται το παρακάτω μενού:*

```
dkoutras@dkoutras-VirtualBox:~/Desktop/PhD/cellularAutomata/cellurarAutomata/task2$ python run
ner.py  -h
Usage: runner.py [options]

Options:
  -h, --help      show this help message and exit
  -r RADIUS       radius of the cycle
  -x X_CENTER     The X center of the cycle
  -y Y_CENTER     The Y center of the cycle
  -s SIZE         The size of the enviroment
  -t TIME_STEPS   The total number of time steps
  -n NOISE_LEVEL  The number of cells with noise
```

Οι παράμετροι RADIUS, X_CENTER, Y_CENTER αναφέρονται στον κύκλο που θα δημιουργήσουμε ενώ το SIZE εκφράζει το μέγεθους του K.A (το οποίο είναι τετραγωνικό), το TIME_STEPS τον αριθμό των βημάτων και το NOISE_LEVEL τον αριθμό των κυψελίδων που εμπεριέχουν θόρυβο (δεδομένου οτι ανεφερόμαστε σε δυαδικά δεδομένα ο θόρυβος ταυτίζεται με την αντίθετη δυαδική τιμή). Με την υλοποίηση που ακολούθησα είναι πολύ εύκολο να τρέξουμε πολλά διαφορετικά πειράματα, κάτι που έκανα και εγώ, ωστόσο για να μην γινω φορτικός στην παρούσα αναφορά επέλεξα να παρουσιάσω το παρακάτω πρόβλημα μόνο:

```
python runner.py -r 15 -x 25 -y 25 -s 50 -t 17 -n 7
```

Υλοποίηση:

Το βασικό κομάτι της υλοποίησης μου είναι η κλάσση *"CellularAutomata"*, η οποία φαίνεται παρακάτω:

```
6   class CellularAutomata:
7
8   ···def __init__(self, rows, columns):
14
15  ···def dist(self, x1, y1, x2, y2):
17
18  ···def drawCycle(self, r, Xcenter, Ycenter):
28
29  ···def printCA(self):
34
35  ···def applyNoiseBoundary(self, noiseLevel = 1):
59
60  ···def applyNoiseRandom(self, noiseLevel = 1):
72
73  ···def plotData(self):
77
78  ···def getData(self):
80
81  ···def getTitle(self):
83
84  ···def saveStep(self, timeStep = None):
90
91  ···def step(self, timeStep = None):
```

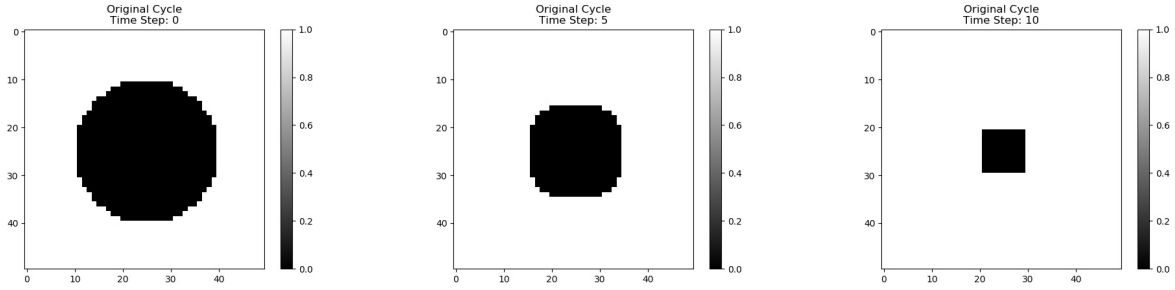
Κατα την δημιουργία του το αντικέιμενο δημιουργεί εναν πίνακα δύο διαστάσεων και θέτει ολες τις τιμές ίσες με την μονάδα (τα προαναφερθέντα υλοποιούνται μέσω του cosntructor “*__init__*”). Στην συνέχει κατα την κλήση της συνάρτησης “*drawCycle(...)*” θέτουμε ορισμένες τιμές ίσες με το μηδέν με τέτοια μορφή ώστες να αποτελούν τον ιδανικό κύκλο που ορίζουμε απο τις παραμέτρους. Εάν θέλουμε να εισάγουμε κάοιον θόρυβο καλούμε τις συναρτήσεις “*applyNoiseBoundary()*” και “*applyNoiseRandom()*”, η πρώτης αλλάζει την τιμή μιας κυψελίδας πάνω στα όρια του κύκλου ενώ η άλλη μέσα στον κύκλο. Επιπλέον, η παράμετρος noiseLevel αποτελεί το πλήθος των κυψελίδων που θα αλλάξουμε προκειμένου να προσωμοιώσουμε τον θόρυβο. Τέλος, στην “*step()*” έχουμε εισάγει τον κανόνα του K.A και όταν καλείται μας προωθεί κατα ένα βήμα στην προσομοίωση. Οι υπόλοιπες συναρτήσεις εκτελούν υποστηρικτικές διαδικασίες.

Πειραματικά Αποτελεσματα

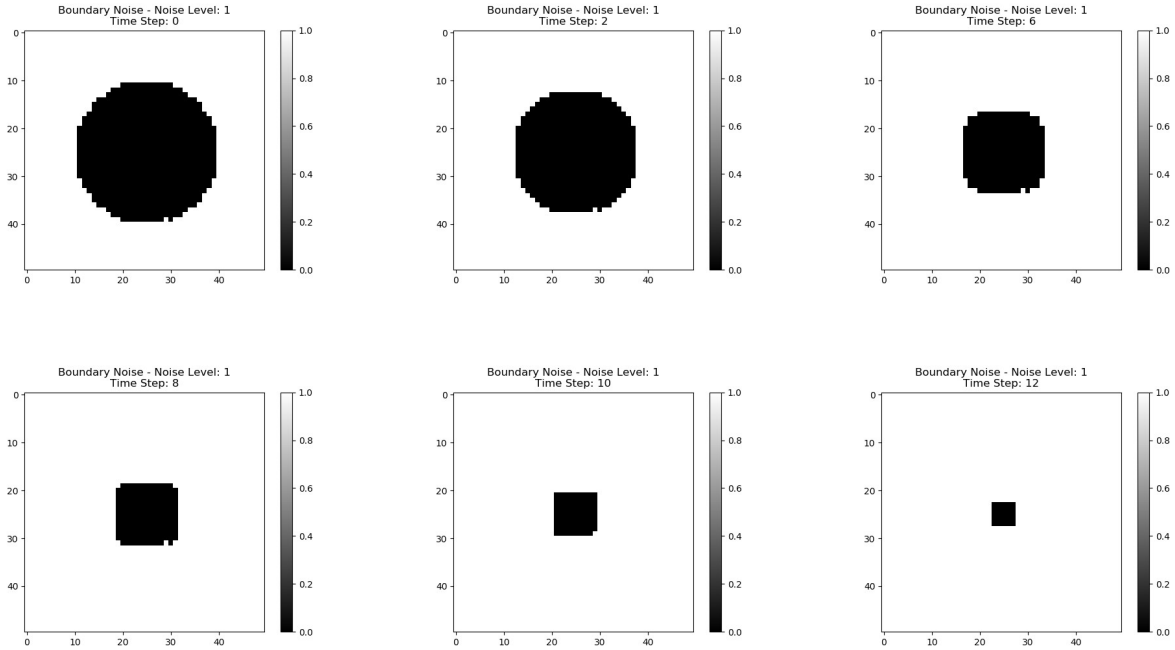
Προκειμένου να καταλάβουμε καλύτερα τον τρόπο λειτουργίας του K.A δημιούργησα 5 διαφορετικές κλάσσεις-περιβάλλοντα. Η πρώτη αποτελεί τον ιδανικό κύκλο (ή για να το θέσω καλύτερα την προσέγγιση του ιδανικού κύκλου), η δεύτερη και η τρίτη εμπεριέχουν θόρυβο στα όρια του κύκλου με το επίπεδο του να είναι 1 και 7 κυψελίδες αντίστοιχα και τέλος η τέταρτη και η πέμπτη κλάσση εμπεριέχουν θόρυβο μέσα στον κύκλο με τα επίδεδα του θορύβου να είναι 1 και 7 αντιστοιχα και πάλι.

Κάθε κλάσση-περιβάλλον αποθηκεύει την κατάσταση της για κάθε χρονική στιγμή καθώς και το πλήθος των μη-μηδενικών στοιχείων.

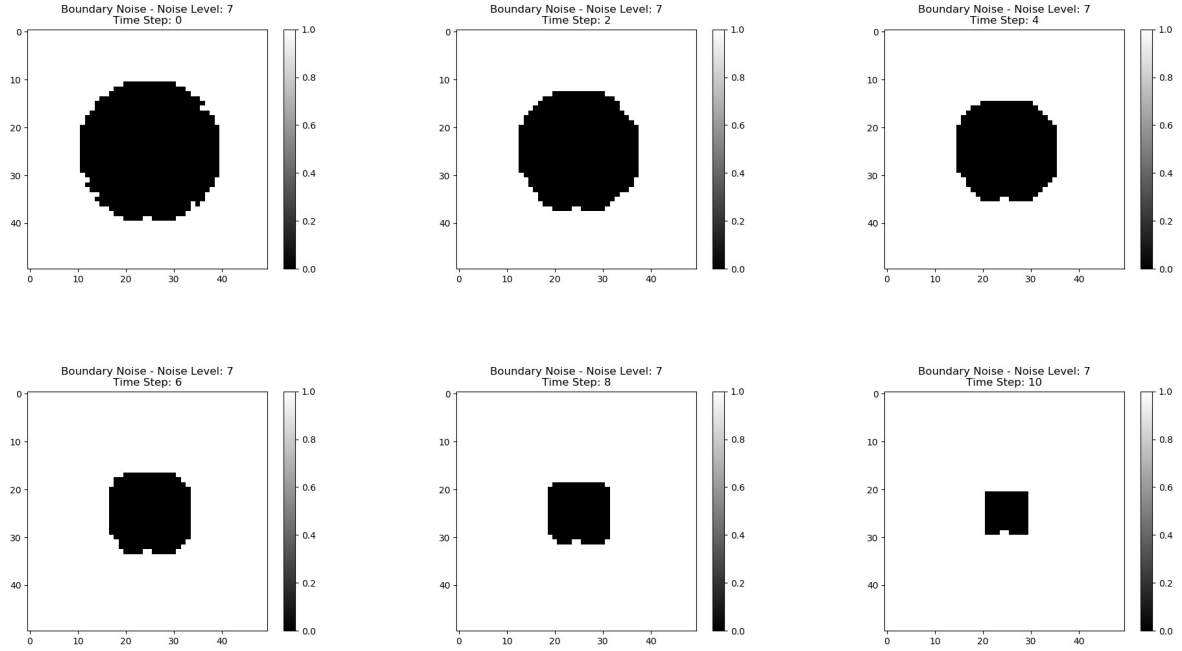
```python
def run2(options):

    rows = options.size
    columns = options.size

    enviroments = [CellularAutomata(rows, columns) for i in range(5)]

    for i in enviroments: i.drawCycle(options.radius, options.X_center, options.Y_center)

    enviroments[1].applyNoiseBoundary()
    enviroments[2].applyNoiseBoundary(noiseLevel = options.noise_level)
    enviroments[3].applyNoiseRandom()
    enviroments[4].applyNoiseRandom(noiseLevel = options.noise_level)

    for i in range(options.time_steps):
        print("Time Step: ",i)
        # env.printCA()
        for env in enviroments: env.step(timeStep=i)

    data = [env.getData() for env in enviroments]
    titles = [env.getTitle() for env in enviroments]

    for index, d in enumerate(data):
        plt.plot(d, label=titles[index])
    plt.legend()
    plt.xlabel('Time Step')
    plt.ylabel('Non Zero Elements [Activated Cells]')
    plt.show()
```



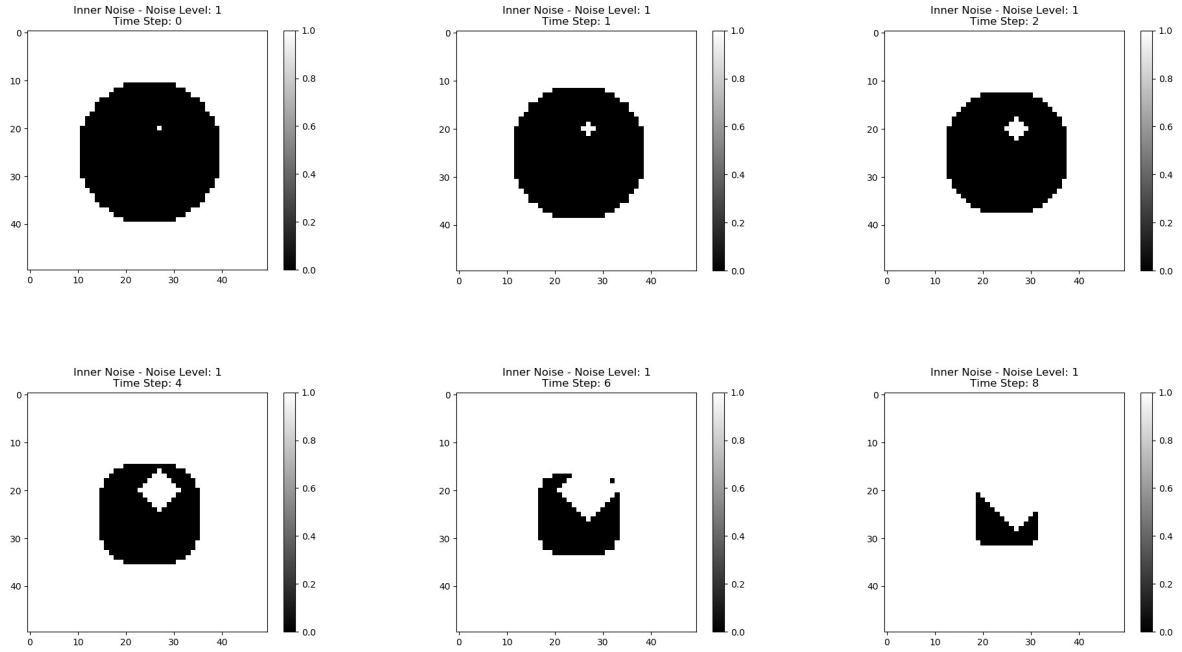
Στην περίπτωση του *“Original Cycle”* έχουμε τον ιδανικό κύκλο και παρατηρούμε πως με την πάροδο του χρόνου σταδιακά μετατρέπεται σε τετράγωνο.



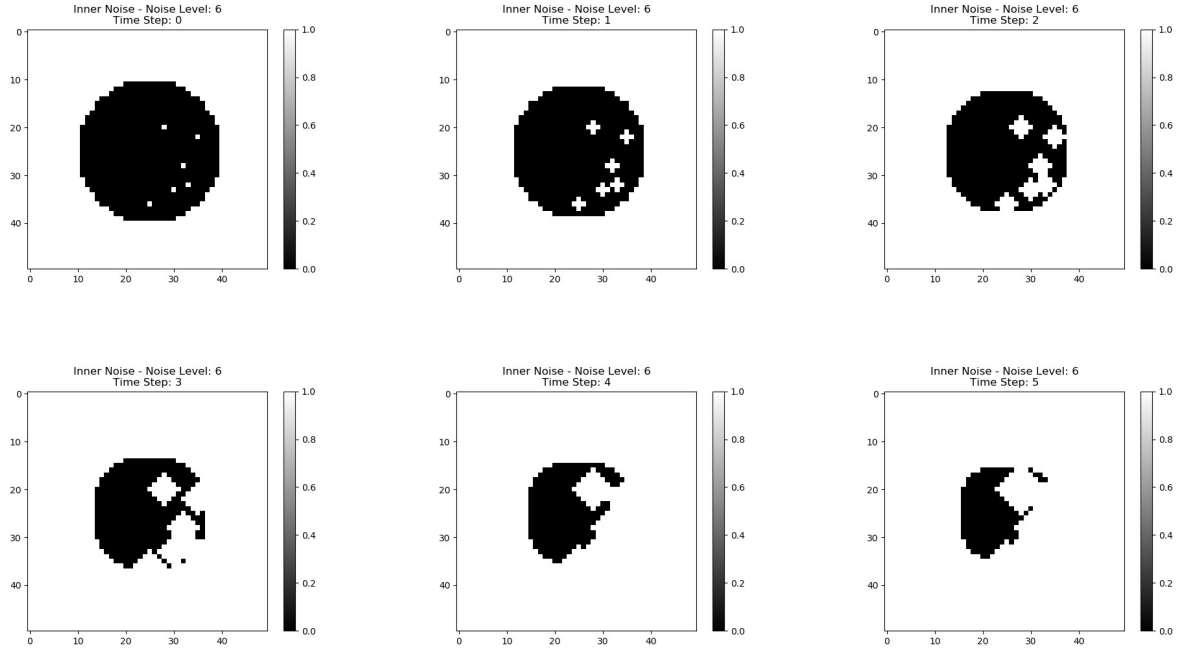
Στην περίπτωη που έχουμε μόνο μια θορυβώδες κυψελίδα στα όρια του κύκλου, παρατηρούμε πως αυτή μεταφέρεται σταδιακα σε όλα τα βήματα αλλα τελικά εξαφανίζεται.



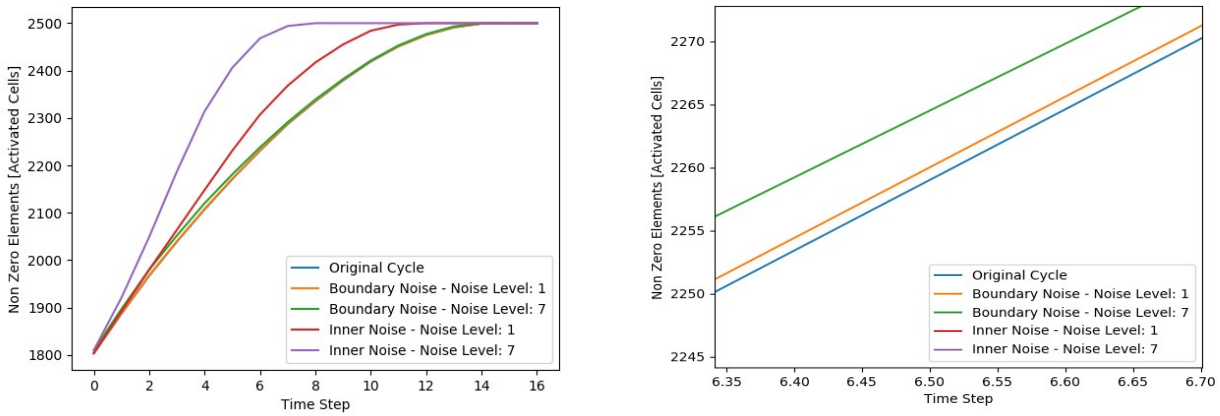
Στην περίπτωση που έχουμε 7 θορυβώδες κυψελίδες στα όρια του κύκλου παρατηρούμε οτι αλλοιώνουν αρκετά την μορφή του σχήματος μας και οτι τα σφάλματα μεταφέρονται στις επόμενες καταστάσεις αλλα και πάλι σταδιακα εξαφανίζονται.



Στην περίπτωση που υπάρχει μια λάθος κυψελίδα μέσα στον κύκλο παρατηρούμε πως αυτή ακομή και σε μικρό διάστημα έχει επηρεάσει κατα πολλύ την όψη του σχήματος.



Αμα το επιπέδο θορύβου αυξηθεί παρατηρούμε οτι η αλλαγή στο σχήμα ειναι ακόμη πιο απότομη.



Το παραπανω διάγραμμα ενεργειας (με τον όρο ενέργεια εννοώ το σύνολο τον στοιχείων που έχουν τιμή ίση με την μονάδα) παρατηρούμε οτι όταν ο θόρυβος βρίσκεται μέσα στον κύκλο το σχημα μας, δηλαδή το Κ.Α, οδηγείτε πιο γρήγορα στην κατάσταση ισοροπίας. Επίσης, όσο πιο μεγάλο είναι το επίπεδο θορύβου τόσο πιο έντονη η κλίση της καμπύλης. Παρατηρώντας το αριστερό διάγραμμα θα μπορούσε να πεί κάποιος ότι ο θόρυβος στις οριακές συνθήκες δεν μπορεί να γίνει αντιληπτός, ωστόσο αμα παρατηρήσουμε καλύτερα το διαγραμμα μας (στα δεξια υπάρχει μια πιο κοντινή οπτική γωνία) θα δούμε οτι υπάρχει διαφορά αλλα είναι πιο δύσκολα αντιληπτή.

Ολοκληρωμένα τα αποτελέσματα μπορείτε να τα βρείτε και στο respository: https://github.com/dimikout3/cellularAutomata/tree/master/cellurarAutomata/task2