Εκφώνηση άσκησης

Το Παιχνίδι της Ζωής (The Game of Life).

Το Project αυτό συνίσταται στην κατασκευή ενός προγράμματος σεναρίου κελύφους, που θα προσομοιώνει την εξέλιξη μίας αποικίας μονοκύτταρων οργανισμών (cells). Η προσομοίωση αυτή είναι γνωστή και ως “Το παιγνίδι της ζωής” ή “Life”.

Η αποικία των cells είναι στην ουσία ένας διδιάστατος πίνακας (π.χ. 20 X 20 θέσεων) που αντιστοιχούν σε θέσεις κελιών μέσα στην αποικία. Το κάθε κελί μπορεί να είναι “εν ζωή” (1) ή όχι (0). Η εξέλιξη της ζωής στην αποικία ξεκινά με ένα συγκεκριμένο “πατρόν” από ζωντανά κελιά. Για παράδειγμα μπορεί κανείς να θέσει τα τέσσερα μεσαία κελιά του πίνακα εν ζωή :

Σε κάθε γενιά (επανάληψη) της εξέλιξης το κάθε κελί του πίνακα αποκτά ζωή ή την χάνει σύμφωνα με τους παρακάτω κανόνες :

1. Αν ένα κενό κελί γειτονεύει με τρία “ζωντανά” κελιά τότε στο κελί αυτό δημιουργείται ζωή.
2. Αν ένα κελί είναι “εν ζωή” και γειτονεύει με 2 ή 3 “ζωντανά” κελιά τότε το κελί διατηρείται
3. Αν ένα κελί είναι “εν ζωή” και γειτονεύει με 1 ή κανένα “ζωντανό” κελί τότε το κελί πεθαίνει λόγω μοναξιάς
4. Αν ένα κελί είναι “εν ζωή” και γειτονεύει με 4 ή περισσότερα “ζωντανά” κελιά τότε το κελί πεθαίνει λόγω συνωστισμού.

Η εξέλιξη συνεχίζεται μέχρι να διακοπεί από τον χρήστη πατώντας ένα πλήκτρο (π.χ. ESCape). Στην προσομοίωση αυτή παρατηρούνται περίεργες συμπεριφορές του πληθυσμού των οργανισμών που εξαρτώνται άμεσα από το αρχικό pattern των ζωντανών κελιών.
Ο αριθμός των κελιών οριζόντια και κάθετα όπως και το αρχικό pattern ζωντανών κελιών πρέπει να καθορίζονται κάθε φορά από τον χρήστη.

Το παιχνίδι της ζωής - στοιχεία

Το παιχνίδι αποτελείται από μια συλλογή κυττάρων τα οποία, βασίζονται σε μερικούς μαθηματικούς κανόνες, μπορούν να ζήσουν, να πεθάνουν ή να πολλαπλασιαστούν. Ανάλογα με τις αρχικές συνθήκες, τα κύτταρα σχηματίζουν διάφορα σχέδια καθ' όλη τη διάρκεια του παιχνιδιού.

Στο Παιχνίδι, της Ζωής (*Game of Life)* ένα παιχνίδι 'κυτταρικών αυτομάτων' για ηλεκτρονικό υπολογιστή, έχουμε ένα πλέγμα από χρωματιστά τετράγωνα που αλλάζουν συνεχώς χρώμα ακολουθώντας κάποιους μηχανικούς κανόνες. Τελικά, αυτό που προκύπτει είναι ένας εντυπωσιακός, συνεχώς μεταβαλλόμενος κόσμος στην οθόνη του υπολογιστή μας. Σε αυτήν εκκολάπτεται συνεχώς κάποιο οικοσύστημα, όπου τα χρώματα δεν σταθεροποιούνται ποτέ και τα 'όντα' αρχίζουν να κολυμπούν στην οθόνη, συμπεριφερόμενα σαν ζωντανές υπάρξεις.

Το παιχνίδι του Conway είναι ένα μαθηματικό θαύμα όπου επαναλαμβανόμενοι και άβουλοι κανόνες από τη μία μπορούν να δημιουργήσουν μία θεαματική, ζωντανή πολυπλοκότητα από την άλλη.

Οι συνέπειες του παιχνιδιού της ζωής είναί εκπληκτικές. Κατά τις επόμενες δύο δεκαετίες, οι φυσικοί ανακάλυψαν ότι διάφορα συστήματα όπως για παράδειγμα ένας σωρός από σπόρους, ο φλοιός της Γης, τα οικοσυστήματα της αλλά ακόμα και οι αγορές της μοιάζουν να λειτουργούν όπως και το παιχνίδι του Conway.

Αυτά τα συστήματα, όπως και πολλά άλλα μοιάζουν να 'αυτοοργανώνονται σε κάτι που το ονομάζουμε ΄'κρίσιμη κατάσταση' μια κατάσταση δηλαδή στην οποία επικρατούν κατά κανόνα διαρκείς μεταβολές και υπερβολική αστάθεια. Οτιδήποτε βρίσκεται, σε κρίσιμη κατάσταση ισορροπεί στην κόψη μιας ξαφνικής, ριζικής αλλαγής και είναι αδύνατο να προβλέψουμε το τι θα συμβεί στη συνεχεία.

Η έννοία της 'αυτο-οργανωνόμενης κρισιμότητας' προτάθηκε για πρώτη φορά το 1987 από τους Per Bak, Chao Tang και Kurt Wiesenfeld. Είναι μια συνολική εξήγηση για τον πολύπλοκο και. ασταθή χαρακτήρα πολλών φαινομένων του κόσμου μας. Μελετώντας τον τρόπο που ξεσπούν οι πυρκαγιές των δασών, τις μαζικές καταστροφές που αλλοιώνουν τη ροή της βιολογικής εξέλιξης ή ακόμα και τον ίδιο το χαρακτήρα της ανθρώπινης ιστορίας, αυτή η έννοία θα μπορούσε να είναί κάτι σαν ένας γενικός νόμος της φύσης που να λέεί ότι. το μέλλον καθορίζεται αναγκαστικά από τελείως απρόβλεπτες ραγδαίες μετατροπές.

Ανάλυση

Το πρόγραμμα μας αποτελείτε από δύο αρχεία. Το αρχείο του πηγαίου αλλά και εκτελέσιμου κώδικα “gol” αλλά και το αρχείο ονόματι “pattern” το οποίο περιέχει ένα έτοιμο πλέγμα προς επεξεργασία και αναπαράσταση. Στην πρώτη γραμμή το “pattern” περιέχει μια κεφαλίδα η οποία αντιπροσωπεύει την διάσταση του πλέγματος. Από την δεύτερη γραμμή και κάτω υπάρχει το “pattern” διαστάσεων NxN (όπου N η κεφαλίδα) το οποίο είναι δυαδικής μορφής (1,0). Για την ανάπτυξη του παιχνιδιού της ζωής θα θεωρήσουμε ως ζωή (ή εν ζωή κελί) την τιμή 1 και ως νεκρή ύλη (ή νεκρό κελί) την τιμή 0.

Για τις ανάγκες του της άσκησης το πρόγραμμα μας θα υλοποιεί δύο περιπτώσεις. Στην πρώτη περίπτωση ο χρήστης θα εισάγει την διάσταση που θέλει να έχει το πλέγμα και το ποσοστό των ζωντανών κελιών που επιθυμεί να έχει το πλέγμα. Στην δεύτερη περίπτωση το πλέγμα θα διαβάζεται από το αρχείο “pattern” το οποίο στην πρώτη σειρά περιέχει την διάσταση του πλέγματος και στις υπόλοιπες, το πλέγμα.

Σε πρώτη φάση θα αναφερθούν όλες οι εντολές με σειρά εμφάνισης στον κώδικα που έχουν χρησιμοποιηθεί με μια μικρή επεξήγηση και στην συνέχεια θα αναλυθεί ο κώδικας καθώς και ο αλγόριθμος του παιχνιδιού της ζωής.

Ανάλυση – Μέρος Α

Εντολές

echo	Εντολή εμφάνισης δεδομένων
read	Διαβάζει απο το πληκτρολόγιο τιμές
clear	Καθαρίζει την οθόνη
while () ;do	Συνθήκη while
done	Όσο (συνθήκη=ΑΛΗΘΗΣ); ΕΚΤΕΛΕΣΕ εντολές ΤΕΛΟΣ_ΕΠΑΝΑΛΗΨΗΣ
if () ; then	Συνθήκη if
elif	ΑΝ(συνθήκη=ΑΛΗΘΗΣ); ΤΟΤΕ

else	εντολές
fi	ΑΛΛΙΩΣ_ΑΝ (συνθήκη=ΑΛΗΘΗΣ); ΤΟΤΕ
	εντολές
	ΑΛΛΙΩΣ
	εντολές
	ΤΕΛΟΣ_ΑΝ
exit	Σταματάει την εκτέλεση του προγράμματος
sed	Φίλτρο επεξεργασίας δεδομένων
let	Πραγματοποιεί αριθμητικές πράξεις
sleep	“παγώνει” το πρόγραμμα για το χρονικό διάστημα που θα δωθεί σαν παράμετρος

Ανάλυση – Μέρος Β

Στο β' μέρος θα αναλύσουμε τον κώδικα μας αλγοριθμικά κομμάτι, κομμάτι.

Κομμάτι 1

```
#!/bin/sh
echo "Welcome to a shell implement of Game of Life"

echo "Menu:"
echo "1. Create your own game of life grid"
echo "2. Implement a ready game of life grid from the pattern file"
echo "3. Exit"
echo "Select by entering from 1 to 3"
read choise
```

Όπως κάθε πρόγραμμα έτσι και το δικό μας στην αρχή εμφανίζει ένα μήνυμά καλώς ορίσματος και στην συνέχεια μας παρουσιάζει το μενού επιλογών του. Οι διαθέσιμες επιλογές μας είναι τρεις και είναι αριθμητικές (1~3). Η αριθμητική μας επιλογή θα αποθηκευτεί στην μεταβλητή choise η οποία είναι και ο κινητήριος μοχλός του προγράμματος μας.

Κομμάτι2

```
while [ $choise -eq 1 -o $choise -eq 2 -o $choise -eq 3 ]; do

if [ $choise -eq 3 ] ; then
exit
else
```

Εδώ ελέγχουμε αν η επιλογή που δώσαμε είναι ορθή. Αν ναι, τότε συνεχίζουμε κανονικά την εκτέλεση ελέγχοντας αν η επιλογή είναι η τρίτη, όπου και το πρόγραμμα θα τερματιστεί. Αλλιώς συνεχίζουμε κανονικά την εκτέλεση.

Κομμάτι 3

```
if [ $choise -eq 2 ] ; then

        read size < pattern

        grid=( `sed '1d' pattern | tr -d '\n' | sed -e 's/1/1 /g' -e 's/0/0 /g' ` )

        let "totalsize =$size*$size"

        agrid=(${grid[@]})
```

Αν η επιλογή μας ήταν η δεύτερη, τότε αποθηκεύουμε στην μεταβλητή “size” πρώτη γραμμή

του αρχείου “pattern” η οποία περιέχει την διάσταση του πλέγματος. Στην συνέχεια, μέσω του φίλτρου sed αποθηκεύουμε σε μορφή μονοδιάστατου πίνακα τα περιεχόμενα του αρχείου “pattern” στον πίνακα “grid”. Έπειτα υπολογίζουμε τον συνολικό αριθμό των κελιών και τον αποθηκεύουμε στην μεταβλητή “totalsize”. Τέλος αντιγράφουμε τα περιεχόμενα του πίνακα grid στον πίνακα agrid. Αυτό γίνεται για να μπορέσουμε επεξεργαστούμε τα δεδομένα μας χωρίς να χρειαστεί να αλλοιώσουμε τα αρχικά.

Επεξήγηση της sed:

sed '1d' pattern | tr -d '\n' | sed -e 's/1/1 /g' -e 's/0/0 /g'

sed '1d' pattern	Εμφάνησε όλο το αρχείο εκτος της πρώτης γραμμής
tr -d '\n'	Σβήσε τον χαρακτήρα νέας γραμμή (ένωσε όλες της γραμμές σε μια)
sed -e 's/1/1 /g' -e 's/0/0 /g'	Αντικατέστησε όπου “1” → “1 “ και όπου “0” → “0 “. Βάλε δηλαδή ένα κενό μετά από κάθε ψηφίο

Κομμάτι 4

```
elif [ $choise -eq 1 ]; then
        echo "Entry dimension"
        read size
        echo "Entry per cent"
        read pers
let "totalsize=$size*$size"

let "cells=($totalsize*$pers*1/100)"
```

Ήμαστε στην πρώτη επιλογή στην οποία ζητάμε απο τον χρήστη να εισάγει την διάσταση του πλέγματος καθώς και το ποσοστό των ζωντανών κελιών του πλέγματος. Στην συνέχεια, υπολογίζουμε τον συνολικό αριθμό των κελιών, καθώς και τα κελιά που θα είναι ζωντανά, βάση του ποσοστού.

Κομμάτι 5

```
while [ $cellcount -lt $totalsize ]; do


        grid[$cellcount]=0
        let "cellcount++"
done
cellcount=0
        ran=$RANDOM
        RANGE=$totalsize
        let "ran %= $RANGE"
        while [ $cellcount -lt $cells ]; do
                grid[$ran]=1
                ran=$RANDOM
                RANGE=$totalsize
                let "ran %= $RANGE"
                echo "Ran" $ran
                let "cellcount++"
        done
agrid=(${grid[@]})


fi
```

Μηδενίζουμε το πλέγμα και στη συνέχεια δημιουργούμε τυχαίους αριθμούς ran οι οποίοι είναι πάντα μικρότεροι του συνολικού αριθμού των κελιών. Έτσι στην τελευταία μας while γεμίζουμε με ζωντανά κελιά το πλέγμα μας, με βάση πάντα το ποσοστό που έχει εισάγει ο χρήστης. Τέλος αντιγράφει τα περιεχόμενα του grid στον agrid

Κομμάτι 6

```
acell=1
```

```
globalcounter=0
while [ $acell -gt 0 ]; do
acell=0
counter=0
while [ $counter -lt $totalsize]; do
clear
```

Κάποια στιγμή κατά την προσομοίωση παύουν να υπάρχουν ζωντανά κελιά στο πλέγμα και τότε τερματίζεται και ο κώδικας μας. Όσο υπάρχουν όμως ζωντανά κελιά, η προσομοίωση συνεχίζεται μέσω της χρήσης της πρώτης while. Στην συνέχεια, με την δεύτερη while, ξεκινά η επεξεργασία του πλέγματος, κελί ανα κελί.

Παρατήρηση

Επειδή στο κέλυφος, δεν υπάρχουν δισ-διάστατοι πίνακες, αναγκαστικά η επεξεργασία του πλέγματος θα γίνει μέσω του μονοδιάστατου πίνακα grid. Σύμφωνα με το game of life για κάθε κελί του πλέγματος μας, θα πρέπει να ελέγχουμε αν τα γειτονικά 8 κελιά είναι ζωντανά ή νεκρά και αναλόγως να εφαρμόσουμε τους κανόνες συνύπαρξης

1	2	3
4		5
6	7	8

Ως δυσ-διάστατος πίνακας, το πλέγμα μας θα είχε την παραπάνω μορφή. Ως μονοδιάστατος όμως θα γινόταν της μορφής:

1	2	3	4		5	6	7	8

Πάμε όμως να δούμε πώς θα εφαρμοζόταν αυτό σε ένα μεγαλύτερο πλέγμα. Έστω πλέγμα με διαστάσεις 5x5.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Το παραπάνω πλέγμα σε μονοδιάστατη μορφή θα ήταν ως εξής:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Αν τώρα θέλουμε να εφαρμόσουμε τους κανόνες του “παιχνιδιού” για το 13ο κελί, θα πρέπει να ελέγξουμε αν τα 8 γειτονικά του κελιά είναι ζωντανά ή νεκρά. Τα κελιά αυτά είναι: 7,8,9,12,14,17,18,19.

1	2	3	4	5	6	**7**	**8**	**9**	10	11	**12**	13	**14**	15	16	**17**	**18**	**19**	20	21	22	23	24	25

Ας μελετήσουμε τις θέσεις των κελιών με βάση την θέση του κελιού 13. Έστω ότι η θέση του 13 είναι I και η διάσταση του πίνακα size.

7	i-size-1
8	i-size

9	i-size+1
12	i-1
14	i+1
17	i+size-1
18	i+size
19	i+size+1

Παρατηρώντας τις παραπάνω θέσεις, μπορούμε πολύ εύκολα να βγάλουμε έναν γενικό αλγόριθμο ελέγχου των γειτονικών κελιών.

Τα παραπάνω όμως ισχύουν μόνο όταν το κελί για το οποίο θα ερευνηθούν οι γείτονες του δεν βρίσκεται στις άκρες του πλέγματος μας. Αν το κελί βρίσκεται στις άκρες του πλέγματος μας, τότε ο αλγόριθμος μας θα αλλάξει.

Έστω ότι το κελί μας βρίσκεται στην θέση 11 του παρακάτω πλέγματος. Τα γειτονικά του κελιά θα είναι τα: 6,7,12,16,17.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Ας το δούμε σε μονοδιάστατη μορφή.

1	2	3	4	5	**6**	**7**	8	9	10	11	**12**	13	14	15	**16**	**17**	18	19	20	21	22	23	24	25

Ας μελετήσουμε ξανά τις θέσεις των κελιών γειτόνων. Έστω πάλι ότι I η θέση του κελιού 11 και size η

διάσταση του πλέγματος.

6	i-size
7	i-size+1
12	i+1
16	i+size
17	i+size+1

Παρατηρούμε ότι η μόνη διαφορά βρίσκεται στο ότι δεν υπάρχουν γειτονικά κελιά στις θέσεις “i-size-1”, “i-1” και “i+size-1”.

Αντιθέτως στην περίπτωση κατα την οποία το κελί μας βρίσκεται στην άκρη του πλέγματος (π.χ. Κελί 15) τα γειτονικά κελιά τα οποία δεν θα υπάρχουν είναι τα: “i-size+1”, “i+1”, “i+size+1”.

Τώρα είμαστε σε θέση να αναπτύξουμε έναν ολοκληρωμένο αλγόριθμο προσπέλασης των γειτονικών κελιών, του κάθε κελιού του πλέγματος μας.

Κομμάτι 7

```
let "beg=$counter%$size"
let "last=($counter+1)%$size"
```

Όταν ένα κελί βρίσκεται στην άκρη αριστερά του πλέγματος πάντα ο αριθμός θέσης του κελιού, θα είναι πολλαπλάσιο της διάστασης του πλέγματος. Στην περίπτωση που βρίσκεται στην άκρη δεξιά του πλέγματος η θέση+1 του θα είναι πολλαπλάσιο της διάστασης του πλέγματος.

Επομένως αν το υπόλοιπο της διαίρεσης της θέσης του κελιού με την διάσταση ισούται με 0, τότε το κελί βρίσκεται στο άκρο αριστερά του πλέγματος.

Ενώ αν το υπόλοιπο της διαίρεσης της θέσης+1 με την διάσταση του πλέγματος ισούται με 0, τότε το κελί θα βρίσκεται στο άκρο δεξιά.

Κομμάτι 8

```
if [ $beg -eq 0 ]; then
                let "side=$counter+1"
                if [ ${grid[$side]} -eq 1 ]; then
                        let "alive++"
                fi
                let "downmid=$counter-$size"
                let "downmidp=$counter-($size-1)"

                let "upmid=$counter+$size"
                let "upp=$counter+($size+1)"

                if [ $downmid -ge 0 -a $downmidp -gt 0 ]; then

                        if [ ${grid[$downmid]} -eq 1 ]; then
                        let "alive++"
                        fi
                        if [ ${grid[$downmidp]} -eq 1 ]; then
                        let "alive++"
                        fi
                fi
                if [ $upmid -lt $totalsize -a $upp -le $totalsize ]; then

                        if [ ${grid[$upmid]} -eq 1 ]; then
                        let "alive++"
                        fi
                        if [ ${grid[$upp]} -eq 1 ]; then
                        let "alive++"
                        fi
                fi
```

Ελέγχουμε αν το κελί μας βρίσκεται στην αρχή του πλέγματος. Αν ναι, υπολογίζουμε τις συντεταγμένες των γειτονικών κελιών και της εκχωρούμε στις μεταβλητές side, downmid, downmidup, upmid, upp. Στη συνέχεια ελέγχουμε αν τα γειτονικά κελιά είναι ζωντανά και αν ναι, αυξήσουμε την μεταβλητή alive. Υπάρχει όμως περίπτωση, τα γειτονικά κελιά να είναι εκτός ορίων του πλέγματος μας. Για αυτό το λόγο ο έλεγχος της κατάστασης των κελιών γίνεται αφού ελεγχθεί η ορθότητα της θέσης αυτών.

Κομμάτι 9

```
elif [ $last -eq 0 ]; then
                let "side=$counter-1"
                if [ ${grid[$side]} -eq 1 ]; then
                        let "alive++"

                fi
                let "downmid=$counter-$size"
                let "downmidp=$counter-($size+1)"

                let "upmid=$counter+$size"
                let "upp=$counter+($size-1)"

                if [ $downmid -gt 0 -a $downmidp -gt 0 ]; then

                        if [ ${grid[$downmid]} -eq 1 ]; then
                        let "alive++"
                        fi
                        if [ ${grid[$downmidp]} -eq 1 ]; then
                        let "alive++"
                        fi
                fi
                if [ $upmid -le $totalsize -a $upp -lt $totalsize ]; then
```

```
                if [ ${grid[$upmid]} -eq 1 ]; then
                let "alive++"
                fi
                if [ ${grid[$upp]} -eq 1 ]; then
                let "alive++"
                fi
        fi
```

Ελέγχουμε αν η θέση του κελιού μας είναι στην άκρη δεξιά του πλέγματος μας και αν ναι υπολογίζουμε τις συντεταγμένες των γειτονικών κελιών και της εκχωρούμε στις μεταβλητές side, downmid, downmidp, upmid, upp. Έπειτα ελέγχουμε την κατάσταση των κελιών αφού πρώτα ελέγξουμε αν οι συντεταγμένες αυτών είναι εντός των ορίων του πλέγματος μας.

Κομμάτι 10

```
else
        let "upmid=$counter+$size"
        let "upup=$counter+$size+1"
        let "updown=$counter+$size-1"
        let "downmid=$counter-$size"
        let "downup=$counter-$size+1"
        let "downdo=$counter-$size-1"
        let "up=$counter+1"
        let "down=$counter-1"
        if [ ${grid[$up]} -eq 1 ]; then
                let "alive++"
                fi
                if [ ${grid[$down]} -eq 1 ]; then
                let "alive++"
```

```
                        fi


        if [ $upup -le $totalsize ]; then
                if [ ${grid[$upmid]} -eq 1 ]; then
                        let "alive++"
                        fi
                        if [ ${grid[$upup]} -eq 1 ]; then
                        let "alive++"
                        fi
                if [ ${grid[$updown]} -eq 1 ]; then
                        let "alive++"
                        fi
        fi
        if [ $downdo -ge 0 ]; then
                        if [ ${grid[$downmid]} -eq 1 ]; then
                        let "alive++"
                        fi
                if [ ${grid[$downup]} -eq 1 ]; then
                        let "alive++"
                        fi
                        if [ ${grid[$downdo]} -eq 1 ]; then
                        let "alive++"
                        fi
        fi
fi
```

Εάν το η θέση του κελιού μας δεν βρίσκεται σε κάποιο από τα αριστερά ή τα δεξιά άκρα του πλέγματος, οι συνθήκες είναι ιδανικές και υπολογίζουμε κανονικά τις διευθύνσεις και των 8 γειτονικών κελιών. Τις διευθύνσεις τις εκχωρούμε στις μεταβλητές upmid, upup, updown, downmid, downup, downdo, up, down. Τέλος ελέγχουμε την κατάσταση των γειτονικών κελιών αφού πρώτα ελέγξουμε αν οι συντεταγμένες των κελιών είναι εντός των ορίων του πλέγματος.

Παρατήρηση

Οι κανόνες του παιχνιδιού της ζωής, είναι οι εξής:

1. Αν ένα κενό κελί γειτονεύει με τρία “ζωντανά” κελιά τότε στο κελί αυτό δημιουργείται ζωή.
2. Αν ένα κελί είναι “εν ζωή” και γειτονεύει με 2 ή 3 “ζωντανά” κελιά τότε το κελί διατηρείται
3. Αν ένα κελί είναι “εν ζωή” και γειτονεύει με 1 ή κανένα “ζωντανό” κελί τότε το κελίπεθαίνει λόγω μοναξιάς
4. Αν ένα κελί είναι “εν ζωή” και γειτονεύει με 4 ή περισσότερα “ζωντανά” κελιά τότε το κελί πεθαίνει λόγω συνωστισμού.

Αν παρατηρήσουμε καλύτερα τους παραπάνω κανόνες, μπορούμε να συμπεράνουμε το εξής:

1. Αν ένα ζωντανό κελί δεν γειτονεύει με 2 ή 3 ζωντανά κελιά, τότε πεθαίνει.
2. Αν ένα νεκρό κελί γειτονεύσει με 3 ζωντανά, τότε το κελί ζωντανεύει.

Έτσι έχουμε απλουστεύσει κατα πολύ τον κώδικα και εφόσον έχουμε μετρήσει τα ζωντανά γειτονικά κελιά μπορούμε να εφαρμόσουμε τους παραπάνω κανόνες.

Συνεχίζοντας..

Κομμάτι 11

```
    if [ ${grid[$counter]} -eq 1 ] ; then


        if [ $alive -eq 2 -o $alive -eq 3 ]; then
            let "agrid[$counter]=1"
        else
```

```
                        let "agrid[$counter]=0"
                fi


        else
                if [ $alive -eq 3 ] ; then
                        let "agrid[$counter]=1"
                fi
        fi
let "counter++"
done
grid=(${agrid[@]})
```

Εφαρμόζουμε τους δύο σύντομους κανόνες που προκύπτουν από τους τέσσερις αρχικούς. Παρατηρήστε ότι οι αλλαγές στην κατάσταση των κελιών εφαρμόζονται στον πίνακα agrid, παρόλο που οι έλεγχοι των κελιών έγιναν στον πίνακα grid. Θυμηθείτε ότι οι δύο πίνακες είναι πανομοιότυποι. Ο λόγος επεξεργασίας του agrid είναι ότι εφόσον οι έλεγχοι γίνονται στον grid, αν επεξεργαστούμε τον grid θα αλλάξουν τα δεδομένα μας, επομένως το αποτέλεσμα δεν θα είναι σωστό. Αφού τελειώσει η επεξεργασία ολόκληρου του πλέγματος, τα περιεχόμενα του πίνακα agrid αντιγράφονται στον πίνακα grid έτσι ώστε στο επόμενο βήμα επεξεργασίας, το αρχικό πλέγμα να είναι το τελευταίο επεξεργασμένο πλέγμα που προέκυψε από την τελευταία επανάληψη- επεξεργασία.

Κομμάτι 12

```
while [ $counter12 -lt $totalsize ] ; do
        while [ $counter11 -lt $size ]; do
                echo -n ${grid[$counter12]}
                if [ ${grid[$counter12]} -eq 1 ]; then
                        let "acell++"
                fi
                let "counter11++"
                let "counter12++"
        done
```

```
counter11=0
        echo
done
echo "Live Cells:" $acell
```

Σε αυτό το κομμάτι κώδικα, εμφανίζουμε το πλέγμα grid μετά την επεξεργασία όλων των κελιών. Παρατηρήστε ότι ο grid είναι ένας μονοδιάστατος πίνακας και εμείς τον εμφανίζουμε σαν δισ-διάστατο. Αυτό γίνεται, εμφανίζοντας κάθε φορά ν στοιχεία του πίνακα και έπειτα, αλλάζοντας γραμμή. Όπου ν ισούται με το size δηλαδή την διάσταση του πίνακα. Κατά την εμφάνισή των κελιών, μετράμε παράλληλα και στη συνέχεια εμφανίζουμε το πλήθος των ζωντανών κελιών που υπάρχουν στο πλέγμα. Αυτό μας χρησιμεύει στον τερματισμό της αναπαράστασης στη περίπτωση κατά την οποία δεν υπάρχουν ζωντανά κελιά στο πλέγμα.

Κομμάτι 13

```
if [ $globalcounter -eq 5 ]; then
        echo "Enter "c" to continue or "b" to break"
        read answer

        if [ $answer =  "b" ]; then
                break
        fi
globalcounter=0
fi
let "globalcounter++"
sleep 1

done
```

Επειδή τις περισσότερες φορές η αναπαράσταση του πλέγματος θα εκτελεστεί πολλές φορές μέχρι να μην υπάρξει κάποιο ζωντανό κελί στο πλέγμα, κάθε 5 αναπαραστάσεις εμφανίζεται κατάλληλο μήνυμα διακοπής ή συνέχιση εκτέλεσης της αναπαράστασης. Η λειτουργία αυτή

επιτυγχάνεται με τον παραπάνω κώδικα. Η εντολή sleep 1 παγώνει για 1 δευτερόλεπτο την οθόνη σε κάθε αναπαράσταση. Αυτό γίνεται γιατί η εμφάνιση και ανανέωση του πλέγματος γίνεται πολύ γρήγορα.

Κομμάτι 14

```
echo
echo
echo "Welcome to a shell implement of Game of Life"

echo "Menu:"
echo "1. Create your own game of life grid"
echo "2. Implement a ready game of life grid from the pattern file"
echo "3. Exit"
echo "Select by entering from 1 to 3"

read choise
fi
done
exit 0
```

Με την ολοκλήρωση της αναπαράστασης ή την διακοπή της, εμφανίζεται το αρχικό μενού επιλογών. Ο κώδικας τερματίζεται όταν ο χρήστης πληκτρολογήσει το 3. Αλλιώς συνεχίζει να εκτελείτε και να αναπαριστά το παιχνίδι της ζωής.

Βιβλιογραφία

- Wikipedia
- Google
- http://tldp.org/LDP/abs/html/