Σχεδιασμός Επαναπρογραμματιζομενο Κυψελωτών Αυτομάτων με Δυναμική Επιλογή Καταστάσεων, Κανόνων και Γειτονιών(Εργασία 7)

Ανάλυση και Σύνθεση Πολύπλοκων Ηλεκτρονικών Συστημάτων

Δημοκρίτειο Πανεπιστήμιο Θράκης

Τμήμα ΗΜΜΥ

Ιούνιος 2025

Παναγιώτα Γροσδουλη

ΑΕΜ: 58523

Διδάσκων Καθηγητής: Γεωργιος Συρακουλης

Περιεχόμενα

1.*Εισαγωγή*

Τα κυψελωτά αυτόματα (cellular automata, CA) είναι μαθηματικά μοντέλα που χρησιμοποιούνται για την προσομοίωση και την ανάλυση δυναμικών συστημάτων με τη χρήση απλών κανόνων εξέλιξης. Αυτά τα συστήματα αποτελούνται από ένα δίκτυο κυψελών τα οποία μπορεί να υπάρχουν σε διάφορους χώρους (μονοδιάστατοι, δισδιάστατοι κ.λπ.). Κάθε κυψέλη του δικτύου έχει μια κατάσταση (ή τιμή) η οποία αλλάζει σε κάθε χρονικό βήμα, βάσει ενός κανόνα που συνήθως εξαρτάται από την κατάσταση της κυψελης αυτής και των γειτονικών της κυψελών.Η διαδικασία αυτή επαναλαμβάνεται διακριτά στο χρόνο, με τις καταστάσεις των κυψελών να εξελίσσονται σύμφωνα με τους καθορισμένους κανόνες. Το χαρακτηριστικό των κυψελωτών αυτομάτων είναι ότι, παρά την απλότητα του κανόνα εξέλιξης, το σύστημα μπορεί να δημιουργεί πολύπλοκα και μη προβλέψιμα πρότυπα και δομές.Ένα από τα πιο διάσημα παραδείγματα κυψελωτών αυτομάτων αποτελεί το Game of Life του Conway, το οποίο λειτουργεί με βάση δύο καταστάσεις για τις κυψελες: "ζωντανές" και "νεκρές" και εξελίσσεται βάσει ενός απλού κανόνα. Παρά την απλότητά του, το Game of Life είναι σε θέση να αναπαράγει πολύπλοκα φαινόμενα όπως αυτογέννεση και τη δημιουργία προτύπων που μοιάζουν με έμβιο οργανισμό.

2. Περιγραφή του Κυψελωτού Αυτομάτου

2.1. Τα Κυψελωτά Αυτόματα (Cellular Automata)

Τα Κυψελωτά Αυτόματα (ΚΑ) αποτελούν υπολογιστικά μοντέλα που βασίζονται σε ένα πλέγμα κυψελίδων, όπου κάθε κυψέλη βρίσκεται σε μία από ένα πεπερασμένο σύνολο καταστάσεων και ενημερώνει την κατάσταση της με βάση τοπικούς κανόνες που εξαρτώνται από τις γειτονικές κυψέλες.

Κάθε Κυψελωτό Αυτόματο ορίζεται από τα εξής βασικά χαρακτηριστικά:

- Πλέγμα (Grid): Ο χώρος στον οποίο ζουν οι κυψέλες. Μπορεί να είναι μονοδιάστατος (1D), δισδιάστατος (2D) ή πολυδιάστατος.
- Καταστάσεις (States): Ο αριθμός των πιθανών καταστάσεων που μπορεί να λάβει κάθε κυψέλη (π.χ. 0 ή 1 σε δυαδικά ΚΑ).
- Γειτονιά (Neighborhood): Το σύνολο των κυψελών που επηρεάζει την ενημέρωση μιας κυψέλης. Π.χ. γειτονιά του Von Neumann ή Moore.
- Κανόνας Ενημέρωσης (Rule): Μια καθορισμένη συνάρτηση που καθορίζει την επόμενη κατάσταση κάθε κυψέλης με βάση τις καταστάσεις των γειτόνων της.

Η εξέλιξη του συστήματος γίνεται σε διακριτά χρονικά βήματα (γενιές), με την κατάσταση όλων των κυψελών να ενημερώνεται ταυτόχρονα, ακολουθώντας τον καθορισμένο κανόνα.Το πιο γνωστό παράδειγμα 1D CA είναι αυτό του Stephen Wolfram, ο οποίος όρισε 256 δυνατούς δυαδικούς κανόνες (0–255) για Κυψελωτά Αυτόματα με δύο καταστάσεις (0,1) και ακτίνα γειτονιάς 1. Παρά τη φαινομενική απλότητά τους, πολλά από αυτά τα Κυψελωτά Αυτόματα παράγουν εξαιρετικά πολύπλοκη και απρόβλεπτη συμπεριφορά.

2.2. Εφαρμογές και Σκοπός της Εργασίας

Τα Κυψελωτά Αυτόματα έχουν εφαρμογές σε διάφορους επιστημονικούς και τεχνολογικούς τομείς. Ενδεικτικά παραδείγματα περιλαμβάνουν:

Φυσική και Βιολογία: Μοντελοποίηση δυναμικών συστημάτων, όπως διάδοση φωτιάς, εξάπλωση ιών, γονιδιακή έκφραση.
Επιστήμη Υπολογιστών: Παράλληλος υπολογισμός, θεωρία υπολογισιμότητας, ανάπτυξη ψευδοτυχαίων αριθμών.
Κρυπτογραφία: Κυψελωτά Αυτόματα ως βάση για αλγόριθμους κρυπτογράφησης, χάρη στην πολυπλοκότητα που παράγουν από απλούς κανόνες.
Γραφικά και Εικονική Πραγματικότητα: Μοντελοποίηση φυσικών φαινομένων όπως νερό, φωτιά και καπνός.
Ο στόχος της εργασίας είναι η ανάπτυξη ενός επαναπρογραμματιζομενο Κυψελωτού Αυτομάτου, δηλαδή ενός CA όπου ο χρήστης θα επιλέγει τον αριθμό των καταστάσεων μετα θα ορίζει το μέγεθος της γειτονιάς (μέσω παραμέτρων όπως ακτίνα),Καθορίζει τον κανόνα εξέλιξης

(σε δεκαδική μορφή),ολοκληρώνοντας να παρακολουθεί την εξέλιξη του συστήματος γραφικά μέσω ενός διαδραστικού περιβάλλοντος. Η εργασία υλοποιείται προγραμματιστικά και περιλαμβάνει τόσο το λογισμικό προσομοίωσης (π.χ. Python/Matplotlib και GUI) όσο και την οπτικοποίηση της συμπεριφοράς του CA για διάφορους κανόνες και παραμέτρους.

3. Περιγραφή της Υλοποίησης

3.1. Επιλογή Γλώσσας και Βιβλιοθηκών

Για την ανάπτυξη της εφαρμογής επιλέχθηκε η γλώσσα προγραμματισμού Python, λόγω της ευκολίας στη σύνταξη κώδικα, της ευρείας χρήσης της σε επιστημονικές και μαθηματικές εφαρμογές, καθώς και της πλούσιας υποστήριξης που προσφέρει σε βιβλιοθήκες σχετικές με υπολογισμούς και οπτικοποίηση.

Συγκεκριμένα, για την υλοποίηση του κυψελωτού αυτομάτου χρησιμοποιήθηκαν οι παρακάτω βιβλιοθήκες:

NumPy: Βιβλιοθήκη για αποδοτική διαχείριση πινάκων και πολυδιάστατων πλεγμάτων, απαραίτητη για την αναπαράσταση και ενημέρωση του πλέγματος καταστάσεων.
Matplotlib: Χρησιμοποιήθηκε για την οπτικοποίηση της εξέλιξης του κυψελωτού αυτομάτου σε μορφή εικόνας, όπου παρουσιάζεται η μεταβολή των καταστάσεων ανά γενιά.
Tkinter: Επιλέχθηκε για τη δημιουργία του γραφικού περιβάλλοντος χρήστη (GUI), το οποίο επιτρέπει την εισαγωγή παραμέτρων και την αλληλεπίδραση με το σύστημα με τρόπο φιλικό προς τον χρήστη.

3.2 Ανάλυση Απαιτήσεων και Σχεδίασης

Η εφαρμογή που αναπτύχθηκε χρησιμοποιεί Κυψελωτά Αυτόματα (Cellular Automata, CA) για την προσομοίωση συστημάτων με μεταβαλλόμενες καταστάσεις κυψελίδων. Η εφαρμογή επιτρέπει στον χρήστη να προσαρμόσει διάφορες παραμέτρους όπως το μέγεθος του πλέγματος, τον αριθμό των καταστάσεων των κυψελίδων, το κατώφλι ενεργοποίησης, και να επιλέξει τους κανόνες εξέλιξης του συστήματος, ώστε να διεξάγει πειράματα με τα δεδομένα που εισάγει.

Αργότερα προστέθηκε και ένα γραφικό περιβάλλον (GUI) για την αλληλεπίδραση του χρήστη με την εφαρμογή. Το περιβάλλον αυτό αναπτύχθηκε και με τη χρήση εξωτερικών βιβλιοθηκών GUI (όπως tkinter, PyQt), αλλά και χρησιμοποιώντας βασικά εργαλεία της Python (π.χ. input(), μέσω απλού rendering σε κονσόλα), επιτρέποντας την αλληλεπίδραση χωρίς εξαρτήσεις.

3.3 Περιγραφή Κώδικα – 1D Επαναπρογραμματιζόμενο Κυψελιδωτό Αυτόματο (CA) Χωρις GUI

Η εφαρμογή επιτρέπει την επιλογή των παραμέτρων μέσω της γραμμής εντολών (CLI) και εκτελεί την προσομοίωση σε μια σειρά γενιών με βάση τις ρυθμίσεις του χρήστη. Στη συνέχεια, τα αποτελέσματα παρουσιάζονται με την χρήση του Matplotlib μέσω γραφημάτων που δείχνουν την εξέλιξη του συστήματος για κάθε γενιά.Ο χρήστης ορίζει το μέγεθος του πλέγματος και τον αριθμό των καταστάσεων των κυψελίδων. Το πλέγμα αρχικοποιείται με τυχαίες τιμές ή με βάση την επιλογή του χρήστη για την αρχική κατάσταση κάθε κυψελίδας.Η εξέλιξη του συστήματος γίνεται με βάση τον κανόνα που επιλέγεται. Ο κανόνας εξαρτάται από τον αριθμό των καταστάσεων που επιλέγει ο χρήστης (π.χ. 2 ή 3 καταστάσεις). Κάθε κυψελίδα εξετάζει τους γείτονές της, και ανανεώνεται με βάση τον κανόνα εξέλιξης.Στην αρχή κάθε γενιάς, το πλέγμα εξελίσσεται με βάση τους γείτονες και τους κανόνες, και τα αποτελέσματα παρουσιάζονται για κάθε γενιά σε ένα διάγραμμα, το οποίο ενημερώνεται δυναμικά.Οι αλλαγές στο πλέγμα απεικονίζονται με τη χρήση matplotlib, το οποίο παρέχει τη δυνατότητα να απεικονίζονται οι καταστάσεις των κυψελίδων σε κάθε γενιά. Ο χρήστης μπορεί να παρακολουθήσει την εξέλιξη του συστήματος και να παρατηρήσει την επίδραση των ρυθμίσεων στις δυναμικές του πλέγματος.

Σύντομη Περιγραφή του Κώδικα

Ο κώδικας περιλαμβάνει τις εξής βασικές λειτουργίες:

initialize_grid(size, states): Δημιουργεί το αρχικό πλέγμα με τυχαίες καταστάσεις για κάθε κυψελίδα, με βάση το μέγεθος του πλέγματος και τον αριθμό καταστάσεων.

evolve_grid(grid, states, threshold): Εκτελεί την εξέλιξη του πλέγματος σύμφωνα με τους κανόνες εξέλιξης που ορίζονται από το κατώφλι ενεργοποίησης των γειτόνων.

visualize_grid(history): Απεικονίζει την πρόοδο της προσομοίωσης σε κάθε γενιά, χρησιμοποιώντας τη βιβλιοθήκη matplotlib για τη δημιουργία γραφημάτων.

Ο κώδικας που αναπτύχθηκε περιλαμβάνει ξεχωριστές εκδοχές Κυψελωτών Αυτομάτων (Cellular Automata - CA). Η πρώτη εκδοχή είναι για 1Δ (1-Dimensional) CA, η δεύτερη εκδοχή είναι για 3Δ (3-Dimensional) Moore neighborhood CA και η τριτη εκδοχη ειναι για 2Δ (2-Dimensional). Θα παραθέσουμε αναλυτική περιγραφή του κώδικα, αναλύοντας τα βήματα και τις λειτουργίες του.

Λειτουργίες Κώδικα

Αρχικοποίηση του Πλέγματος (initialize_grid και initialize_grid_3d)

1Δ ΚΑ :Το πλέγμα αρχικοποιείται ως ένας πίνακας με 0 σε όλες τις θέσεις εκτός από τη μεσαία θέση, που παίρνει την τιμή 1 (ενεργή κυψελίδα).Στην περίπτωση της 3Δ προσομοίωσης, το πλέγμα είναι τρισδιάστατο και οι κυψελίδες τυχαία λαμβάνουν τιμές από το εύρος [0, states).
Υπολογισμός Πίνακα Κανόνων (generate_rule_table):Εδώ ορίζεται η εξέλιξη του συστήματος με βάση έναν κανόνα που παρέχεται από τον χρήστη (σε δεκαδική μορφή).

Η συνάρτηση generate_rule_table:Υπολογίζει τον πίνακα κανόνων, ο οποίος περιλαμβάνει τις μετατροπές των καταστάσεων των κυψελίδων σύμφωνα με τον κανόνα που έχει εισαχθεί.Χρησιμοποιεί τον αριθμό του κανόνα για να τον μετατρέψει σε δυαδική μορφή, και στη συνέχεια τον αναστρέφει για να το χρησιμοποιήσει στον πίνακα κανόνων.
Η είσοδος της συνάρτησης είναι ο αριθμός του κανόνα και οι καταστάσεις, ενώ εξάγει έναν πίνακα που καθορίζει την κατάσταση κάθε κυψελίδας βάσει των γειτόνων της.
Εξέλιξη του Συστήματος (evolve_1d και evolve_grid)

1Δ ΚΑ:Για κάθε κυψελίδα στο πλέγμα, η συνάρτηση evolve_1d εξετάζει τις γειτονικές της κυψελίδες (σύμφωνα με τον ορισμένο ακτίνα) και υπολογίζει τη νέα της κατάσταση με βάση τον πίνακα κανόνων.Η γειτονιά γύρω από κάθε κυψελίδα δημιουργείται μέσω της εξετάσεως των στοιχείων του πίνακα γειτονιών, και ο υπολογισμός του κλειδιού γίνεται με έναν σταθμισμένο συνδυασμό των καταστάσεων των γειτονικών κυψελίδων.
3Δ ΚΑ:
Η συνάρτηση evolve_grid εξετάζει κάθε κυψελίδα σε ένα τρισδιάστατο πλέγμα. Χρησιμοποιεί τη γειτονιά Moore 3D, που σημαίνει ότι μια κυψελίδα έχει 26 γείτονες σε έναν τρισδιάστατο χώρο.
Η εξέλιξη βασίζεται στο κατώφλι ενεργοποίησης: εάν ο αριθμός των ενεργών γειτόνων ξεπερνά το κατώφλι, η κυψελίδα εξελίσσεται σε μία νέα κατάσταση.

Οπτικοποίηση Αποτελεσμάτων (Matplotlib)Χρησιμοποιούμε την matplotlib για την απεικόνιση της εξέλιξης του πλέγματος σε κάθε γενιά.
Στην περίπτωση του 1Δ ΚΑ, το πλέγμα απεικονίζεται με την χρήση της συνάρτησης plt.imshow, η οποία δημιουργεί μία εικόνα του πλέγματος για κάθε γενιά.Στην περίπτωση του 3Δ ΚΑ, το

πλέγμα απεικονίζεται σε μια τομή του κεντρικού επιπέδου του πλέγματος για κάθε γενιά.

Ο χρήστης καθορίζει
Αριθμός καταστάσεων κυψελίδων: Πόσες καταστάσεις μπορεί να έχει κάθε κυψελίδα.
Μέγεθος πλέγματος: Ο αριθμός των κυψελίδων σε κάθε διάσταση του πλέγματος (π.χ., 50 για ένα πλέγμα 50x50 για 1Δ).
Αριθμός γενιών: Ο αριθμός των γενιών που θα εξελιχθούν.
Κανόνας ή Κατώφλι Ενεργοποίησης: Ο κανόνας για την εξέλιξη ή το κατώφλι για την 3Δ προσομοίωση.

Η αλληλεπίδραση των κυψελίδων είναι κεντρική στον τρόπο λειτουργίας του συστήματος
Στην περίπτωση των 1Δ ΚΑ, κάθε κυψελίδα επηρεάζεται από τις κυψελίδες γύρω της (1 ή 2 γείτονες), οι οποίες αποφασίζουν την επόμενη κατάσταση της.
Στην περίπτωση των 3Δ ΚΑ, οι κυψελίδες επηρεάζονται από τους 26 γείτονές τους στον τρισδιάστατο χώρο (Moore neighborhood), και η κατάσταση τους εξελίσσεται με βάση τις ενεργές γειτονικές κυψελίδες.
Αυτή η δυναμική αλληλεπίδραση μεταξύ των κυψελίδων είναι εκείνη που προκαλεί την εξέλιξη του συστήματος μέσα σε πολλές γενιές, αναπτύσσοντας συχνά ενδιαφέροντα μοτίβα και συμπεριφορές.

Περιγραφή Κώδικα – 1D Επαναπρογραμματιζόμενο Κυψελωτό Αυτόματο (χωρίς GUI)

Η υλοποίηση του μονοδιάστατου Κυψελωτού Αυτομάτου (1D CA) πραγματοποιήθηκε σε γλώσσα Python, αξιοποιώντας τις βιβλιοθήκες NumPy για αριθμητικούς υπολογισμούς και Matplotlib για την οπτικοποίηση της εξέλιξης του πλέγματος.

3.1 Εισαγωγή Παραμέτρων από τον Χρήστη

Αρχικά, μέσω εντολών input(), ο χρήστης ορίζει τις βασικές παραμέτρους του συστήματος:

Αριθμός καταστάσεων (num_states): Ορίζεται ο αριθμός των καταστάσεων που μπορεί να λάβει κάθε κυψέλη. Για παράδειγμα, 2 για δυαδικό CA.
Μέγεθος πλέγματος (grid_size): Το μήκος της μονοδιάστατης διάταξης κυψελών.
Αριθμός γενιών (generations): Το πλήθος των διακριτών βημάτων εξέλιξης που θα εκτελεστούν.
Κανόνας εξέλιξης (rule_base): Ο κανόνας του CA σε δεκαδική μορφή (π.χ. 30, 110), που καθορίζει πώς αλλάζουν οι καταστάσεις των κυψελών.

3.2 Δημιουργία Πίνακα Κανόνων (generate_rule_table)

Η συνάρτηση generate_rule_table μετατρέπει τον αριθμό του κανόνα από τη δεκαδική μορφή σε μια ακολουθία συμβόλων (σε βάση ίση με τον αριθμό των καταστάσεων). Από αυτή την ακολουθία παράγεται ένας πίνακας (dictionary) κανόνων, όπου για κάθε πιθανό μοτίβο γειτονιάς αντιστοιχεί η νέα κατάσταση της κεντρικής κυψέλης.

Η ακτίνα της γειτονιάς είναι προεπιλεγμένη στο 1, δηλαδή εξετάζονται η ίδια η κυψέλη και οι άμεσοι γείτονές της αριστερά και δεξιά, άρα το μέγεθος της γειτονιάς είναι 3 κυψέλες.

3.3 Αρχικοποίηση Πλέγματος (initialize_grid)

Η συνάρτηση initialize_grid δημιουργεί το αρχικό μονοδιάστατο πλέγμα, όπου όλες οι κυψέλες ξεκινούν στην κατάσταση 0, εκτός από την κυψέλη που βρίσκεται στο κέντρο, η οποία αρχικοποιείται στην κατάσταση 1. Αυτό αποτελεί μια τυπική αρχική συνθήκη που επιτρέπει την παρακολούθηση της εξέλιξης του CA από ένα απλό αρχικό μοτίβο.

3.4 Εξέλιξη Κυψελωτού Αυτομάτου (evolve_1d)

Η βασική συνάρτηση εξέλιξης evolve_1d λαμβάνει το τρέχον πλέγμα και τον πίνακα κανόνων, και υπολογίζει το επόμενο πλέγμα. Για κάθε κυψέλη, δημιουργείται η γειτονιά της (συμπεριλαμβάνοντας και τα άκρα με χρήση περιοδικής συνθήκης), μετατρέπεται σε αριθμητικό κλειδί, και με βάση αυτό επιλέγεται η νέα κατάσταση από τον πίνακα κανόνων.

3.5 Εκτέλεση και Οπτικοποίηση

Η εξέλιξη του πλέγματος επαναλαμβάνεται για τον αριθμό γενιών που έχει οριστεί. Όλες οι διαδοχικές καταστάσεις αποθηκεύονται σε λίστα history, η οποία στη συνέχεια απεικονίζεται με χρήση της matplotlib.pyplot.imshow. Η απεικόνιση εμφανίζει την εξέλιξη των καταστάσεων σε έναν χρωματικό χάρτη, όπου ο άξονας x αντιστοιχεί στις κυψέλες και ο άξονας y στις γενιές.

Η επιλογή των χρωμάτων γίνεται με το colormap 'viridis', ενώ το γράφημα περιλαμβάνει κατάλληλους τίτλους και ετικέτες για την καλύτερη κατανόηση των αποτελεσμάτων.

Ο Κώδικας ακολουθεί παρακάτω:

```
import numpy as np

import matplotlib.pyplot as plt


# === ΧΡΗΣΤΗΣ ΟΡΙΖΕΙ ===

num_states = int(input("Αριθμός καταστάσεων (π.χ. 2): "))

grid_size = int(input("Μέγεθος πλέγματος (π.χ. 50): "))

generations = int(input("Αριθμός γενιών (π.χ. 50): "))

rule_base = int(input("Εισάγετε τον κανόνα (σε δεκαδική μορφή π.χ. 30, 110): "))
```

```
# === ΥΠΟΛΟΓΙΣΜΟΣ RULETABLE ===
def generate_rule_table(rule_number, states, radius=1):
    num_neigh = (2 * radius + 1)  # Ο αριθμός των γειτόνων που επηρεάζουν την κυψέλη
    total_configs = states ** num_neigh  # Ο συνολικός αριθμός συνδυασμών
    rule_bin = np.base_repr(rule_number, base=states).zfill(total_configs)  # Ο κανόνας σε δυαδική μορφή
    rule_table = {i: int(rule_bin[::-1][i]) for i in range(total_configs)}  # Δημιουργία πίνακα κανόνων
    return rule_table

# === ΑΡΧΙΚΟΠΟΙΗΣΗ ===
def initialize_grid(size):
    grid = np.zeros(size, dtype=int)
    grid[size // 2] = 1  # Μοναδική ενεργή κυψελίδα στη μέση
    return grid

# === ΕΞΕΛΙΞΗ 1D ΚΑ ===
def evolve_1d(grid, rule_table, states, radius=1):
    size = len(grid)
    new_grid = np.zeros_like(grid)
    for i in range(size):
        neighborhood = []
        # Δημιουργία της γειτονιάς (γύρω από την κυψέλη)
        for j in range(i - radius, i + radius + 1):
            idx = j % size
            neighborhood.append(grid[idx])
        key = 0
        # Υπολογισμός του κλειδιού για τον πίνακα κανόνων
        for k, val in enumerate(neighborhood):
            key += val * (states ** (len(neighborhood) - k - 1))
        new_grid[i] = rule_table[key]
```

```
    return new_grid

# === ΕΚΤΕΛΕΣΗ ===
rule_number = int(rule_base)
rule_table = generate_rule_table(rule_number, num_states)
current_grid = initialize_grid(grid_size)

history = [current_grid.copy()]
for _ in range(generations - 1):
    current_grid = evolve_1d(current_grid, rule_table, num_states)
    history.append(current_grid.copy())

# === ΟΠΤΙΚΟΠΟΙΗΣΗ ===
plt.imshow(history, cmap='viridis', interpolation='nearest')
plt.title(f"Programmable 1D CA - Rule {rule_number} - {num_states} states")
plt.xlabel("Κυψελίδες")
plt.ylabel("Γενιές")
plt.colorbar(label="Κατάσταση")
plt.show()
```

Παράδειγμα Εκτέλεσης

Για παράδειγμα, αν ο χρήστης εισάγει τις παρακάτω τιμές:

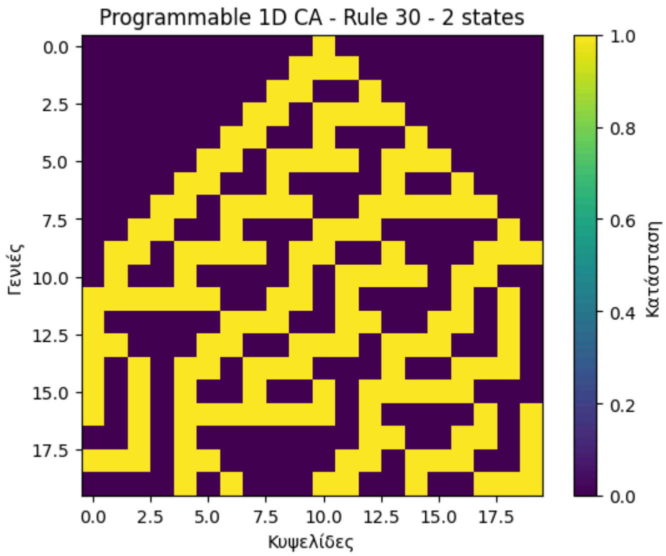
Αριθμός καταστάσεων (π.χ. 2): 2

Μέγεθος πλέγματος (π.χ. 50): 20

Αριθμός γενιών (π.χ. 50): 20

Εισάγετε τον κανόνα (σε δεκαδική μορφή π.χ. 30, 110): 30

Το πρόγραμμα θα εξελίξει έναν δυαδικό μονοδιάστατο CA με 20 κυψέλες και 20 γενιές, βάσει του κανόνα 30, εμφανίζοντας το τελικό αποτέλεσμα ως χρωματική απεικόνιση της δυναμικής συμπεριφοράς.



Σχήμα 1:1D-CA Χωρις GUI

Το Σχήμα 1.1 παρουσιάζει την εξέλιξη ενός μονοδιάστατου κυτταρικού αυτόματου με βάση τον Κανόνα 30 και δύο καταστάσεις. Παρατηρείται η δημιουργία χαοτικών προτύπων από απλή αρχική κατάσταση, γεγονός που καθιστά τον κανόνα 30 χρήσιμο σε εφαρμογές όπως η ψευδοτυχαία παραγωγή. Η απεικόνιση χωρίς χρήση GUI αποδεικνύει τη δυνατότητα προγραμματιστικού ελέγχου και ανάλυσης τέτοιων συστημάτων μέσω κατάλληλων εργαλείων.

*4.*Παράδειγμα Υλοποίησης με GUI για 1D Κυψελωτό Αυτόματο

Στην προσπάθεια βελτίωσης της διαδραστικότητας της εφαρμογής, υλοποιήθηκε ένα γραφικό περιβάλλον χρήστη (GUI) με χρήση της βιβλιοθήκης Tkinter της Python. Το GUI επιτρέπει στον χρήστη να εισάγει εύκολα τις παραμέτρους του κυψελωτού αυτόματου, χωρίς να απαιτείται τροποποίηση του κώδικα ή εκτέλεση σε περιβάλλον κονσόλας.

4.1 Περιγραφή Λειτουργικότητας

Το γραφικό περιβάλλον περιλαμβάνει πεδία εισαγωγής για:

Αριθμό καταστάσεων: Ορίζει το πλήθος των πιθανών καταστάσεων κάθε κυψέλης.
Μέγεθος πλέγματος: Το μήκος της μονοδιάστατης σειράς κυψελών.
Αριθμό γενιών: Το πλήθος των βημάτων εξέλιξης.
Κανόνα εξέλιξης: Ο αριθμός του κανόνα σε δεκαδική μορφή (π.χ. 30, 110).
Ο χρήστης συμπληρώνει τα πεδία και πατά το κουμπί "Εκτέλεση", το οποίο ενεργοποιεί τη λειτουργία run_simulation().

4.2 Περιγραφή Κώδικα

Η συνάρτηση run_simulation: Διαβάζει τις τιμές που έδωσε ο χρήστης.
Δημιουργεί τον πίνακα κανόνων καλώντας τη generate_rule_table.
Αρχικοποιεί το πλέγμα με τη initialize_grid.Υπολογίζει την εξέλιξη του CA για τον αριθμό των γενιών που ορίστηκαν.Οπτικοποιεί τα αποτελέσματα μέσω της matplotlib, εμφανίζοντας τη δυναμική εξέλιξη του συστήματος.Σε περίπτωση μη έγκυρης εισαγωγής (π.χ. γράμματα αντί για αριθμούς), το πρόγραμμα εμφανίζει κατάλληλο μήνυμα λάθους μέσω messagebox.

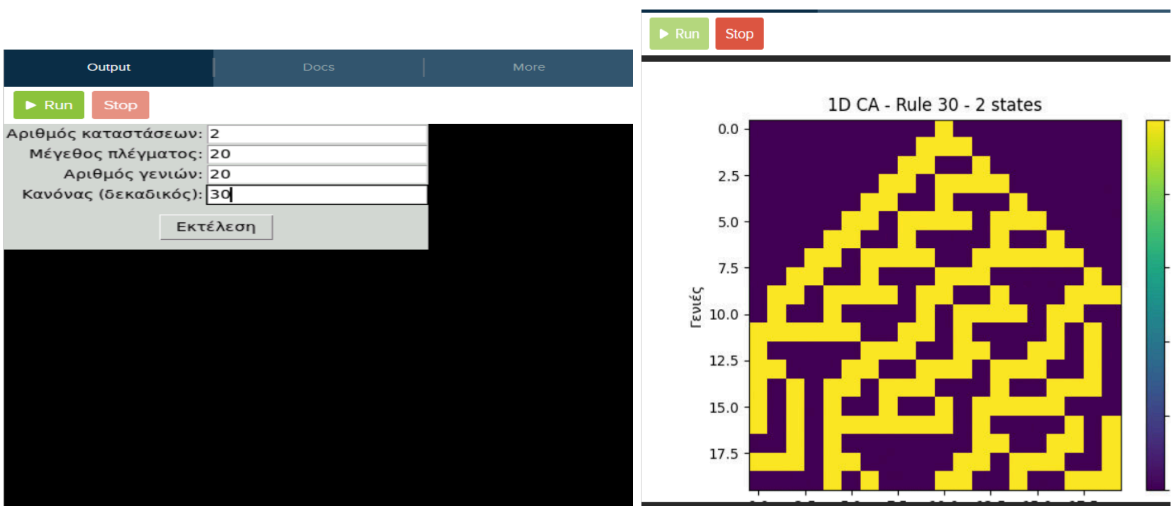
Παράδειγμα του κώδικα με την χρήση του gui για 1D,χρησιμοποιήθηκε η online σελίδα https://codehs.com/sandbox/id/python-graphics-tkinter-IPnUun?filepath=main.py

```
import numpy as np

import matplotlib.pyplot as plt

import tkinter as tk

from tkinter import messagebox


# === ΥΠΟΛΟΓΙΣΜΟΣ RULETABLE ===

def generate_rule_table(rule_number, states, radius=1):

    num_neigh = (2 * radius + 1)

    total_configs = states ** num_neigh
```

```
    rule_bin = np.base_repr(rule_number, base=states).zfill(total_configs)
    rule_table = {i: int(rule_bin[::-1][i]) for i in range(total_configs)}
    return rule_table

# === ΑΡΧΙΚΟΠΟΙΗΣΗ ===
def initialize_grid(size):
    grid = np.zeros(size, dtype=int)
    grid[size // 2] = 1
    return grid

# === ΕΞΕΛΙΞΗ 1D ΚΑ ===
def evolve_1d(grid, rule_table, states, radius=1):
    size = len(grid)
    new_grid = np.zeros_like(grid)
    for i in range(size):
        neighborhood = []
        for j in range(i - radius, i + radius + 1):
            idx = j % size
            neighborhood.append(grid[idx])
        key = 0
        for k, val in enumerate(neighborhood):
            key += val * (states ** (len(neighborhood) - k - 1))
        new_grid[i] = rule_table[key]
    return new_grid

# === ΕΚΤΕΛΕΣΗ ΚΑΙ ΠΡΟΒΟΛΗ ===
def run_simulation():
    try:
        num_states = int(entry_states.get())
        grid_size = int(entry_grid.get())
```

```
        generations = int(entry_generations.get())
        rule_number = int(entry_rule.get())

        rule_table = generate_rule_table(rule_number, num_states)
        current_grid = initialize_grid(grid_size)

        history = [current_grid.copy()]
        for _ in range(generations - 1):
            current_grid = evolve_1d(current_grid, rule_table, num_states)
            history.append(current_grid.copy())

        plt.imshow(history, cmap='viridis', interpolation='nearest')
        plt.title(f"1D CA - Rule {rule_number} - {num_states} states")
        plt.xlabel("Κυψελίδες")
        plt.ylabel("Γενιές")
        plt.colorbar(label="Κατάσταση")
        plt.show()

    except ValueError:
        messagebox.showerror("Σφάλμα", "Παρακαλώ εισάγετε έγκυρους ακέραιους αριθμούς.")

# === GUI ===
root = tk.Tk()
root.title("Προσομοίωση Κυψελωτών Αυτομάτων (1D CA)")

tk.Label(root, text="Αριθμός καταστάσεων:").grid(row=0, column=0, sticky="e")
entry_states = tk.Entry(root)
entry_states.grid(row=0, column=1)

tk.Label(root, text="Μέγεθος πλέγματος:").grid(row=1, column=0, sticky="e")
```

```
entry_grid = tk.Entry(root)
entry_grid.grid(row=1, column=1)

tk.Label(root, text="Αριθμός γενιών:").grid(row=2, column=0, sticky="e")
entry_generations = tk.Entry(root)
entry_generations.grid(row=2, column=1)

tk.Label(root, text="Κανόνας (δεκαδικός):").grid(row=3, column=0, sticky="e")
entry_rule = tk.Entry(root)
entry_rule.grid(row=3, column=1)

tk.Button(root, text="Εκτέλεση", command=run_simulation).grid(row=4, column=0, columnspan=2, pady=10)

root.mainloop()
```



Σχήμα 2:1D-CA με GUI

Παράδειγμα Εκτέλεσης

Για παράδειγμα, αν ο χρήστης εισάγει τις παρακάτω τιμές:

Αριθμός καταστάσεων (π.χ. 2): 2

Μέγεθος πλέγματος (π.χ. 50): 20

Αριθμός γενιών (π.χ. 50): 20

Εισάγετε τον κανόνα (σε δεκαδική μορφή π.χ. 30, 110): 30

5. Υλοποίηση Τρισδιάστατου Κυψελωτού Αυτομάτου (3D CA) χωρίς GUI

Στην υλοποίηση αυτή παρουσιάζεται ένα τρισδιάστατο κυψελωτό αυτόματο, το οποίο εξελίσσεται σε έναν κύβο διαστάσεων size×size×size, όπου κάθε κυψέλη μπορεί να λάβει μία από τις states δυνατές καταστάσεις.

5.1 Παράμετροι Εισόδου

Ο χρήστης καλείται να ορίσει

Μέγεθος πλέγματος: Το μήκος της κάθε διάστασης του κύβου.
Αριθμό γενιών: Τον αριθμό των εξελικτικών βημάτων του συστήματος.
Αριθμό καταστάσεων: Τις πιθανές καταστάσεις που μπορεί να έχει κάθε κυψέλη.
Κατώφλι ενεργοποίησης: Το όριο γειτόνων που καθορίζει πότε μια κυψέλη θα αλλάξει κατάσταση.

5.2 Αρχικοποίηση Πλέγματος

Η τρισδιάστατη μήτρα αρχικοποιείται τυχαία με καταστάσεις από το [0,states−1]. Η τυχαία κατανομή προσδίδει ποικιλία στην αρχική κατάσταση, επιτρέποντας διαφορετικά σενάρια εξέλιξης.

5.3 Γειτονιά Moore 3D

Η γειτονιά Moore ορίζεται ως το σύνολο των 26 κυψελών που περιβάλλουν μια συγκεκριμένη κυψέλη σε όλες τις τρεις διαστάσεις (οι άξονες x, y, z). Η εξέταση των γειτόνων γίνεται με κυκλική περιτύλιξη (periodic boundary conditions), ώστε να εξασφαλίζεται ότι το πλέγμα είναι κλειστό και δεν υπάρχουν άκρα.

5.4 Κανόνας Εξέλιξης

Ο κανόνας που εφαρμόζεται καθορίζει την αλλαγή της κατάστασης κάθε κυψέλης ανάλογα με το πλήθος των ενεργών (κατάσταση μεγαλύτερη του μηδενός) γειτόνων της.Συγκεκριμένα,αν ο αριθμός ενεργών γειτόνων είναι ίσος ή μεγαλύτερος από το κατώφλι ενεργοποίησης, τότε η κατάσταση της κυψέλης αυξάνεται κατά 1 modulo του αριθμού των καταστάσεων.
Διαφορετικά, η κατάσταση παραμένει ίδια.
Αυτή η λογική επιτρέπει την προσομοίωση δυναμικών φαινομένων όπου η “ενεργοποίηση” μιας περιοχής εξαρτάται από την πυκνότητα ενεργών κυψελών γύρω της.

5.5 Οπτικοποίηση

Παρά την απουσία γραφικού περιβάλλοντος χρήστη (GUI), γίνεται οπτικοποίηση της εξέλιξης μέσω του matplotlib, προβάλλοντας κάθε φορά την τομή του πλέγματος στο μέσο της τρίτης διάστασης (επίπεδο Z = size//2). Αυτή η 2D τομή ανανεώνεται κάθε γενιά, δίνοντας μια αίσθηση της δυναμικής εξέλιξης του 3D CA.

Κώδικας χωρίς GUI

```
import numpy as np
import matplotlib.pyplot as plt

# === ΠΑΡΑΜΕΤΡΟΙ ===
size = int(input("Μέγεθος πλέγματος (π.χ. 10): "))
generations = int(input("Αριθμός γενιών (π.χ. 20): "))
states = int(input("Αριθμός καταστάσεων κυψελίδων (π.χ. 2): "))
threshold = int(input("Κατώφλι ενεργοποίησης (π.χ. 13 για Moore 3D): "))

# === ΑΡΧΙΚΟΠΟΙΗΣΗ ===
def initialize_grid_3d(size, states):
    grid = np.random.randint(0, states, (size, size, size))
    return grid

# === Moore 3D ΓΕΙΤΟΝΙΑ ===
def get_moore_neighbors_3d(grid, x, y, z):
    neighbors = []
    size = grid.shape[0]
    for dx in [-1, 0, 1]:
        for dy in [-1, 0, 1]:
            for dz in [-1, 0, 1]:
                if dx == 0 and dy == 0 and dz == 0:
                    continue
                nx, ny, nz = (x + dx) % size, (y + dy) % size, (z + dz) % size
                neighbors.append(grid[nx, ny, nz])
    return neighbors
```

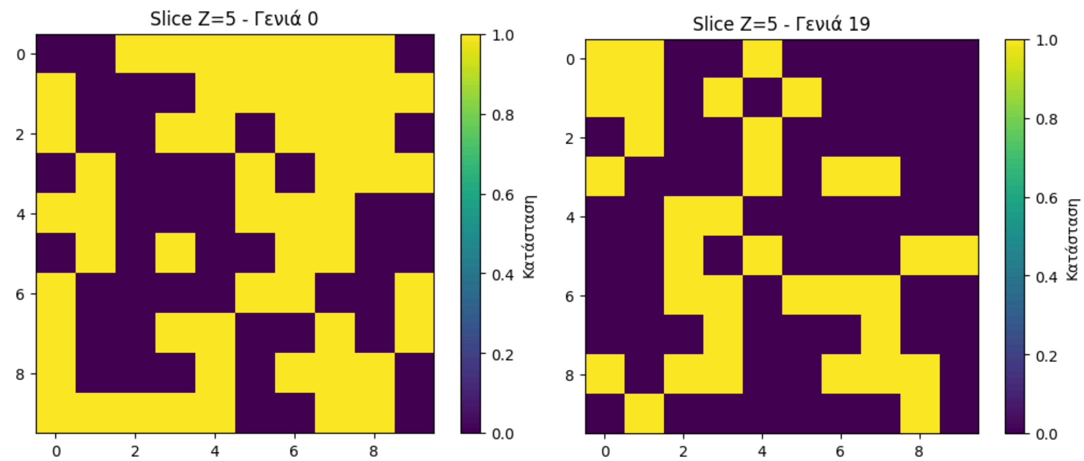
```python
# === ΚΑΝΟΝΑΣ ΕΞΕΛΙΞΗΣ ===
def evolve_grid(grid, states, threshold):
    new_grid = np.zeros_like(grid)
    size = grid.shape[0]
    for x in range(size):
        for y in range(size):
            for z in range(size):
                neighbors = get_moore_neighbors_3d(grid, x, y, z)
                active_neighbors = sum(1 for n in neighbors if n > 0)
                if active_neighbors >= threshold:
                    new_grid[x, y, z] = (grid[x, y, z] + 1) % states
                else:
                    new_grid[x, y, z] = grid[x, y, z]
    return new_grid

# === ΕΚΤΕΛΕΣΗ ===
grid = initialize_grid_3d(size, states)

for t in range(generations):
    print(f"Γενιά {t}")
    mid_slice = grid[:, :, size // 2]
    plt.imshow(mid_slice, cmap="viridis")
    plt.title(f"Slice Z={size//2} - Γενιά {t}")
    plt.colorbar(label="Κατάσταση")
    plt.pause(0.3)
    grid = evolve_grid(grid, states, threshold)

plt.show()
```

Μέγεθος πλέγματος (π.χ. 10): 10

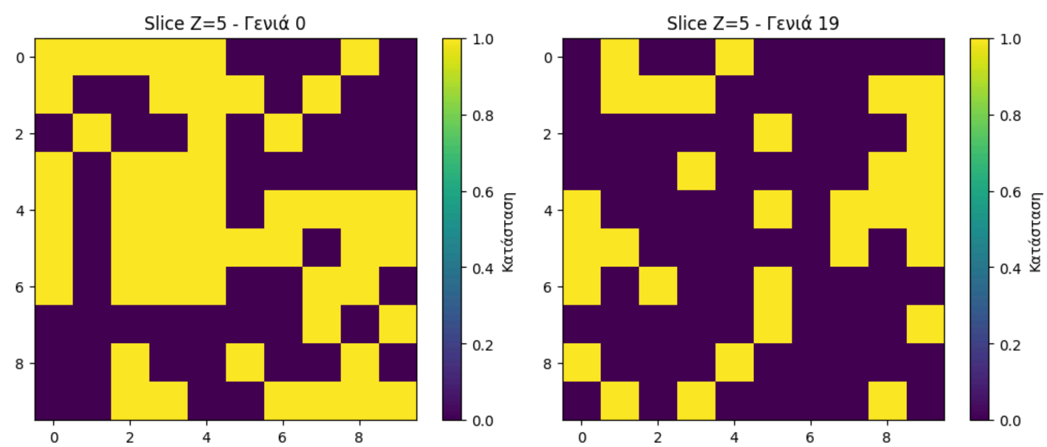
Αριθμός γενιών (π.χ. 20): 20

Αριθμός καταστάσεων κυψελίδων (π.χ. 2): 2

Κατώφλι ενεργοποίησης (π.χ. 13 για Moore 3D): 13



Σχήμα 3:3D-CA χωρις GUI



Σχήμα 4:3D-CA χωρις GUI

Το παραπάνω σχήμα απεικονίζει την εξέλιξη ενός τρισδιάστατου κυτταρικού αυτόματου με διαστάσεις πλέγματος 10×10×10, 20 γενιές και δύο δυνατές καταστάσεις κυψελίδων. Παρουσιάζονται δύο οριζόντιες τομές (slice Z=5) από την αρχική γενιά (0) και την τελική (19). Η ενεργοποίηση των κυψελίδων καθορίζεται από κατώφλι 13 στον γειτονικό Moore 3D κανόνα, ο οποίος λαμβάνει υπόψη 26 γειτονικές κυψέλες.Παρατηρείται ότι από μια σχετικά τυχαία αρχική κατανομή, το σύστημα οδηγείται σε μια πιο οργανωμένη και τοπικά σταθερή διαμόρφωση. Το κατώφλι ενεργοποίησης επηρεάζει σημαντικά τη συμπεριφορά του μοντέλου, περιορίζοντας την υπερβολική εξάπλωση ή εξαφάνιση των ενεργών κυψελίδων. Οι διαφορές μεταξύ της αρχικής και της τελικής κατάστασης καταδεικνύουν την επίδραση των κανόνων εξέλιξης στο συνολικό πρότυπο του πλέγματος.Το συγκεκριμένο παράδειγμα υπογραμμίζει τη δυνατότητα των 3D-CA να προσομοιώνουν πολύπλοκα χωροχρονικά φαινόμενα, όπως εξάπλωση, φιλτράρισμα ή τοπική συσσώρευση, με ευρεία εφαρμογή σε φυσικά συστήματα και υπολογιστικά μοντέλα.

Πειραματισμοί με 3Δ Κυψελωτους Αυτόματους

Σε αυτό το κεφάλαιο, εξετάζουμε τη συμπεριφορά των κυψελωτών αυτομάτων σε τρεις διαστάσεις (3D). Η προσθήκη της διάστασης του βάθους αυξάνει σημαντικά την πολυπλοκότητα του συστήματος και μας επιτρέπει να παρατηρήσουμε πιο σύνθετα και δυναμικά φαινόμενα. Αντί να έχουμε απλές αλληλεπιδράσεις σε δύο διαστάσεις, προσθέτουμε και την τρίτη διάσταση, η οποία επηρεάζει τη συμπεριφορά και τις αλληλεπιδράσεις των κυψελίδων.
Ακολουθούν οι πειραματισμοί που πραγματοποιήθηκαν για το 3D σύστημα:

Πειραματισμός 1 (Απλό Σύστημα)
Περιγραφή:
Μέγεθος Πλέγματος: 20x20
Γενιές: 50
Κατάσταση Κυψελίδων: 2 (ενεργό ή ανενεργό)
Κατώφλι Ενεργοποίησης: 3

Εδώ υπάρχει ένα απλό σύστημα με μικρό μέγεθος πλέγματος, δύο καταστάσεις για τις κυψελίδες και 50 γενιές για την εξέλιξη. Ο στόχος είναι να παρατηρήσουμε τη βασική δυναμική του κυψελωτού αυτόματου με έναν πολύ απλό κανόνα. Εξαιτίας του μικρού μεγέθους πλέγματος, οι αλληλεπιδράσεις θα είναι περιορισμένες, και το σύστημα θα εξελιχθεί γρήγορα. Αναμένονται απλά μοτίβα που επαναλαμβάνονται ή σταθεροποιούνται με την πάροδο του χρόνου.
Το κατώφλι ενεργοποίησης 3 είναι αρκετά χαμηλό για να παρατηρήσουμε ενδιαφέροντα μοτίβα χωρίς να γίνει υπερβολικά περίπλοκο το σύστημα.

Κώδικας
Ο κώδικας αποτελεί τον ίδιο για κάθε περίπτωση,αλλάζουν το μέγεθος πλέγματος,γενιές,κατάσταση κυψελίδων και κατώφλι ενεργοποίησης.

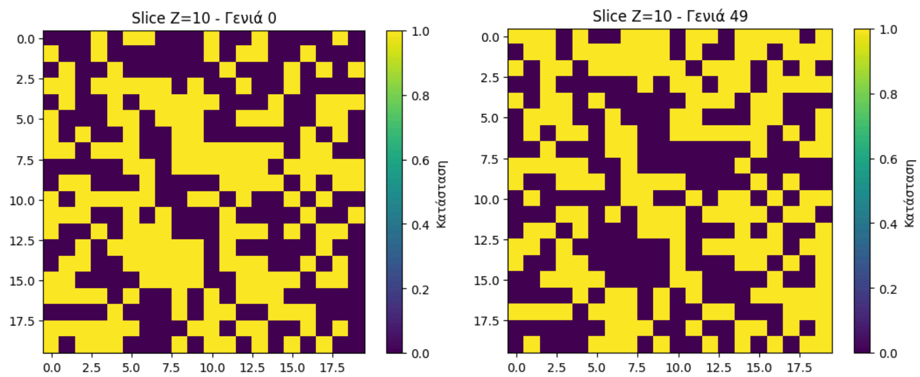
```
import numpy as np

import matplotlib.pyplot as plt


# === ΠΑΡΑΜΕΤΡΟΙ ===
```

```python
size = int(input("Μέγεθος πλέγματος (π.χ. 10): "))
generations = int(input("Αριθμός γενιών (π.χ. 20): "))
states = int(input("Αριθμός καταστάσεων κυψελίδων (π.χ. 2): "))
threshold = int(input("Κατώφλι ενεργοποίησης (π.χ. 13 για Moore 3D): "))

# === ΑΡΧΙΚΟΠΟΙΗΣΗ ===
def initialize_grid_3d(size, states):
    grid = np.random.randint(0, states, (size, size, size))
    return grid

# === Moore 3D ΓΕΙΤΟΝΙΑ ===
def get_moore_neighbors_3d(grid, x, y, z):
    neighbors = []
    size = grid.shape[0]
    for dx in [-1, 0, 1]:
        for dy in [-1, 0, 1]:
            for dz in [-1, 0, 1]:
                if dx == 0 and dy == 0 and dz == 0:
                    continue
                nx, ny, nz = (x + dx) % size, (y + dy) % size, (z + dz) % size
                neighbors.append(grid[nx, ny, nz])
    return neighbors

# === ΚΑΝΟΝΑΣ ΕΞΕΛΙΞΗΣ ===
def evolve_grid(grid, states, threshold):
    new_grid = np.zeros_like(grid)
    size = grid.shape[0]
    for x in range(size):
```

```
        for y in range(size):
            for z in range(size):
                neighbors = get_moore_neighbors_3d(grid, x, y, z)
                active_neighbors = sum(1 for n in neighbors if n > 0)
                if active_neighbors >= threshold:
                    new_grid[x, y, z] = (grid[x, y, z] + 1) % states
                else:
                    new_grid[x, y, z] = grid[x, y, z]
    return new_grid

# === ΕΚΤΕΛΕΣΗ ===
grid = initialize_grid_3d(size, states)

for t in range(generations):
    print(f"Γενιά {t}")
    mid_slice = grid[:, :, size // 2]
    plt.imshow(mid_slice, cmap="viridis")
    plt.title(f"Slice Z={size//2} - Γενιά {t}")
    plt.colorbar(label="Κατάσταση")
    plt.pause(0.3)  # Παύση για animation effect
    plt.clf()  # Καθαρισμός του καμβά για την επόμενη γενιά
    grid = evolve_grid(grid, states, threshold)

plt.show()  # Εμφάνιση του τελικού αποτελέσματος
```

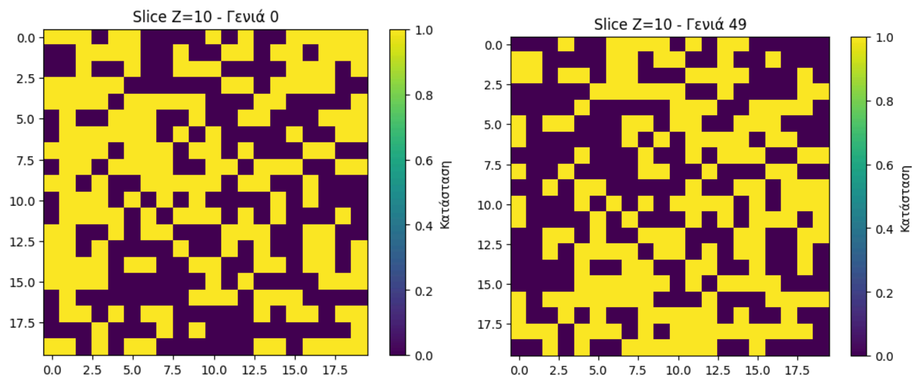
Στη συνέχεια παρουσιάζονται με τον παραπάνω κώδικα και τις ίδιες μεταβλητές δυο διαφορετικες εκδοχες των κυψελωτών αυτόματων

1ο τρέξιμο



Σχήμα 5:3D-CA χωρις GUI

2ο τρέξιμο



Σχήμα 6:3D-CA χωρις GUI

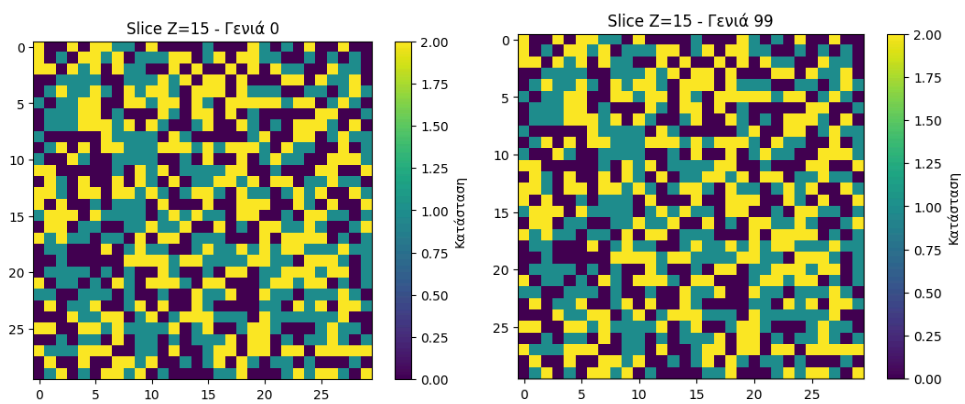
Ο Πειραματισμός 1 αφορά την προσομοίωση ενός απλού τρισδιάστατου κυτταρικού αυτόματου με διαστάσεις πλέγματος 20×20×20, 50 γενιές, 2 καταστάσεις κυψελίδων και κατώφλι ενεργοποίησης 3. Στην εικόνα παρουσιάζονται δύο ανεξάρτητα τρεξίματα του ίδιου συστήματος, με τομές (slice Z=10) από την αρχική (Γενιά 0) και την τελική (Γενιά 49) κατάσταση.Παρατηρείται ότι και στα δύο τρεξίματα, το αρχικό τυχαίο μοτίβο κυψελίδων διατηρεί έντονη ποικιλομορφία και δεν συγκλίνει σε κάποια σταθερή ή επαναλαμβανόμενη κατάσταση. Η χρήση χαμηλού κατωφλίου ενεργοποίησης (3) επιτρέπει την εύκολη «επιβίωση» και ενεργοποίηση κυψελίδων, οδηγώντας σε δυναμικά αλλά μη προβλέψιμα μοτίβα.Η σύγκριση των δύο τρεξιμάτων αναδεικνύει την ευαισθησία του συστήματος στις αρχικές συνθήκες, γεγονός που είναι χαρακτηριστικό σε πολλά μη-γραμμικά συστήματα. Παρότι οι κανόνες είναι ίδιοι, οι τελικές κατανομές παρουσιάζουν διαφορές, υποδηλώνοντας χαοτική συμπεριφορά.
Συμπερασματικά, το συγκεκριμένο απλό μοντέλο CA δείχνει ότι ακόμα και με βασικούς κανόνες και παραμέτρους, το σύστημα μπορεί να παράγει πολύπλοκες και διαφορετικές εξελίξεις.

Πειραματισμός 2 (Πιο Πολύπλοκο Σύστημα)

Περιγραφή:
Μέγεθος Πλέγματος: 30x30
Γενιές: 100
Κατάσταση Κυψελίδων: 3
Κατώφλι Ενεργοποίησης: 5

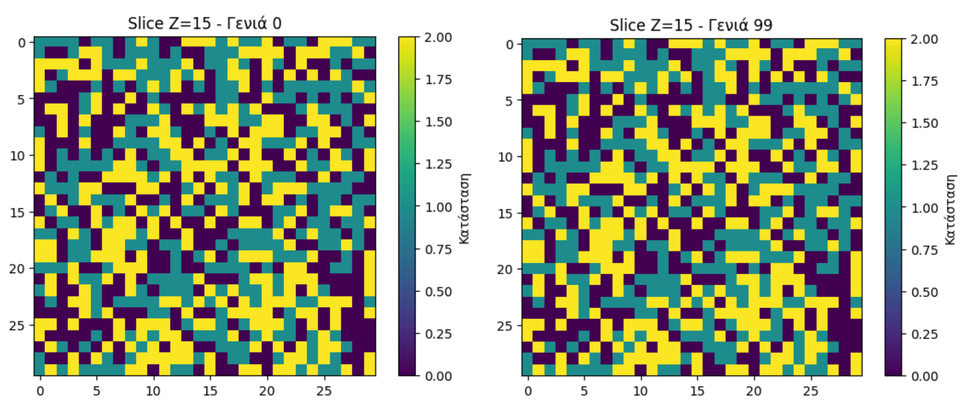
Ανάλυση
Το μεγαλύτερο πλέγμα (30x30) ενισχύει την πολυπλοκότητα του συστήματος, επιτρέποντας πιο ενδιαφέροντα μοτίβα και αλληλεπιδράσεις μεταξύ των κυψελίδων.Η προσθήκη της τρίτης κατάστασης δίνει περισσότερες δυνατότητες για σύνθετες αλληλεπιδράσεις και αλλαγές κατάστασης. Το κατώφλι ενεργοποίησης 5 απαιτεί μεγαλύτερη αλληλεπίδραση μεταξύ γειτόνων για να αλλάξει η κατάσταση μιας κυψελίδας. Αυτό σημαίνει ότι το σύστημα θα παρουσιάσει λιγότερη συχνότητα αλλαγών και πιθανώς πιο περίπλοκες διακυμάνσεις στην εξέλιξή του.Με 100 γενιές, το σύστημα θα έχει αρκετό χρόνο για να αναπτύξει πιο περίπλοκες δομές ή ακόμα και να καταρρεύσει σε τυχαία καταστάσεις.

1ο τρέξιμο



Σχήμα 7:1D-CA χώρις GUI

2ο τρέξιμο



Σχήμα 8:3D-CA χωρίς GUI

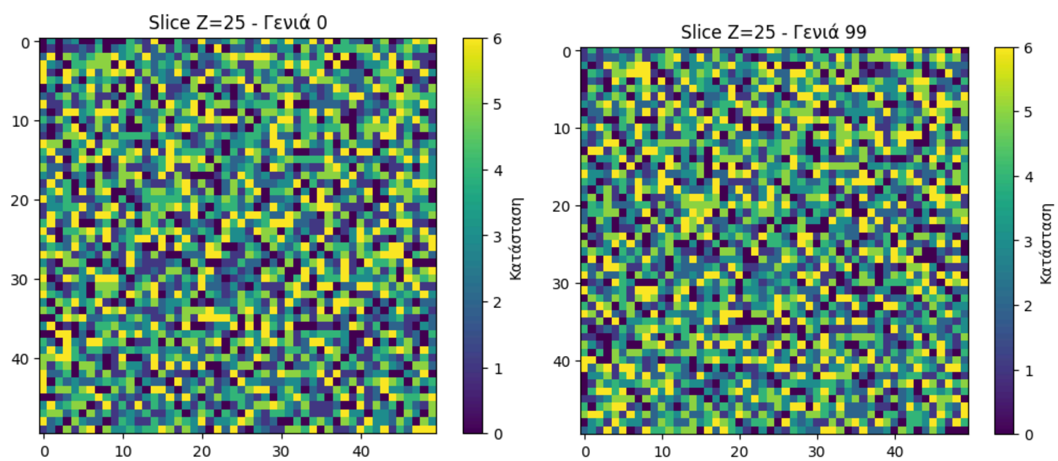
Αποτελέσματα:

Αρχικά πραγματοποιήθηκαν δύο ανεξάρτητα τρεξίματα σε πλέγμα 30×30, με 3 καταστάσεις κυψελίδων και κατώφλι ενεργοποίησης ίσο με 5, για 100 γενιές. Και στις δύο περιπτώσεις παρατηρήθηκε η διατήρηση της χωρικής ετερογένειας, χωρίς να προκύψει καθολική επικράτηση κάποιας κατάστασης. Η δυναμική του συστήματος φαίνεται να οδηγεί είτε σε περιοδικές κατανομές είτε σε τοπικές συστάδες που σταθεροποιούνται, υποδεικνύοντας πιθανή ύπαρξη πολλών σημείων ισορροπίας. Το γεγονός ότι τα δύο τρεξίματα κατέληξαν σε διαφορετικές κατανομές ενισχύει τον ρόλο της αρχικής τυχαιότητας στην εξέλιξη του συστήματος. Συνολικά, το μοντέλο παρουσιάζει ανθεκτικότητα στην ομογενοποίηση και διατηρεί πλούσια δυναμική συμπεριφορά.

Πειραματισμός 3 (Μεγαλύτερο Πλέγμα και Πιο Πολύπλοκες Καταστάσεις)

Περιγραφή:
Μέγεθος Πλέγματος: 50x50
Γενιές: 100
Κατάσταση Κυψελίδων: 7
Κατώφλι Ενεργοποίησης: 7

Ανάλυση:
Το μεγαλύτερο πλέγμα (50x50) με 100 γενιές επιτρέπει την εμφάνιση πολύπλοκων μοτίβων και τη μελέτη τους σε μεγαλύτερη κλίμακα. Θα είναι δύσκολο να παρατηρήσουμε την εξέλιξη ολόκληρου του συστήματος με μία μόνο παρατήρηση, αλλά θα εμφανιστούν μεγαλύτερες κλίμακες συμπεριφοράς.Η αύξηση των καταστάσεων σε 7 προσφέρει περισσότερες δυνατότητες για αλλαγές στην κατάσταση των κυψελίδων, οδηγώντας σε πιο περίπλοκες αλληλεπιδράσεις.Το κατώφλι 7 απαιτεί ακόμη μεγαλύτερη αλληλεπίδραση για να αλλάξει η κατάσταση μιας κυψελίδας, γεγονός που καθιστά το σύστημα πιο ασταθές και προδιαθέτει για τη δημιουργία πολύπλοκων μοτίβων.Οι παράμετροι αυτές καθιστούν το σύστημα πιο ευαίσθητο και ανοικτό σε απρόβλεπτες ή τυχαίες καταστάσεις, κάτι που το καθιστά δύσκολο να προβλεφθεί και να αναλυθεί πλήρως.



Σχήμα 9:3D-CA χωρίς GUI

Συμπεράσματα

Η μελέτη και προσομοίωση κυψελωτών αυτομάτων σε 1D, 2D και 3D περιβάλλοντα κατέδειξε τη σημαντική επίδραση των παραμέτρων (μέγεθος πλέγματος, αριθμός καταστάσεων, κατώφλι ενεργοποίησης, τύπος γειτονιάς και διαστάσεις) στη δυναμική συμπεριφορά του συστήματος.

Κατά τη διάρκεια των πειραμάτων που πραγματοποιήθηκαν, έγινε φανερό ότι ακόμη και μικρές τροποποιήσεις στις παραμέτρους μπορούν να οδηγήσουν σε ριζικά διαφορετικές συμπεριφορές, από πλήρως προβλέψιμα και περιοδικά μοτίβα μέχρι μη γραμμικές και χαοτικές εξελίξεις.

7.Υλοποίησης με GUI για 3D Κυψελωτό Αυτόματο

Στην προσπάθεια βελτίωσης της διαδραστικότητας και ευχρηστίας της προσομοίωσης, υλοποιήθηκε γραφικό περιβάλλον χρήστη (GUI) με χρήση της βιβλιοθήκης Tkinter της Python. Το περιβάλλον αυτό επιτρέπει στον χρήστη να εισάγει εύκολα τις παραμέτρους του τρισδιάστατου κυψελωτού αυτόματου χωρίς να απαιτείται τροποποίηση του κώδικα ή χρήση της κονσόλας.

7.1 Περιγραφή Λειτουργικότητας

Το GUI περιλαμβάνει πεδία εισαγωγής για τις βασικές παραμέτρους:

Μέγεθος πλέγματος: Ορίζει το μήκος κάθε διάστασης του τρισδιάστατου κύβου κυψελών (π.χ. 10 για πλέγμα 10×10×10).
Αριθμός γενιών: Ο αριθμός των χρονικών βημάτων εξέλιξης του συστήματος.
Αριθμός καταστάσεων κυψελών: Ο αριθμός των διαφορετικών καταστάσεων που μπορεί να λάβει κάθε κυψέλη.
Κατώφλι ενεργοποίησης: Το πλήθος ενεργών γειτόνων (κατάσταση > 0) που απαιτούνται ώστε μια κυψέλη να αλλάξει κατάσταση στην επόμενη γενιά.
Ο χρήστης συμπληρώνει τα πεδία και πατά το κουμπί "Εκκίνηση", το οποίο καλεί τη συνάρτηση run_simulation().

7.2 Περιγραφή Κώδικα

Η συνάρτηση run_simulation() λειτουργεί ως εξής:Διαβάζει και ελέγχει τις τιμές που έχει εισάγει ο χρήστης στα πεδία. Σε περίπτωση μη έγκυρης εισαγωγής (π.χ. χαρακτήρες αντί για αριθμούς), εμφανίζει μήνυμα σφάλματος μέσω του messagebox και διακόπτει τη λειτουργία.
Καλεί τη συνάρτηση initialize_grid_3d() για να δημιουργήσει το αρχικό τρισδιάστατο πλέγμα κυψελών με τυχαίες καταστάσεις.Εκτελεί σε βρόχο την εξέλιξη του κυψελωτού αυτόματου με βάση τον κανόνα εξέλιξης, που εφαρμόζεται μέσω της συνάρτησης evolve_grid(). Ο κανόνας λαμβάνει υπόψη τη γειτονιά Moore 3D και το κατώφλι ενεργοποίησης.Σε κάθε γενιά, εξάγει και απεικονίζει οπτικά τη μεσαία τομή του πλέγματος κατά τη διάσταση z, χρησιμοποιώντας τη βιβλιοθήκη matplotlib. Η απεικόνιση ενημερώνεται δυναμικά με σκοπό τη ζωντανή παρακολούθηση της εξέλιξης.

Με αυτόν τον τρόπο, ο χρήστης μπορεί εύκολα να πειραματιστεί με διαφορετικές παραμέτρους, να παρατηρήσει τη χωρική και χρονική συμπεριφορά του συστήματος και να μελετήσει τις επιδράσεις των ρυθμίσεων στην εξέλιξη του τρισδιάστατου κυψελωτού αυτόματου.

κώδικας με την χρήση Gui για 3D

Χρησιμοποιήθηκε η Online σελίδα:https://codehs.com/sandbox/id/python-graphics-tkinter-IPnUun?filepath=main.py

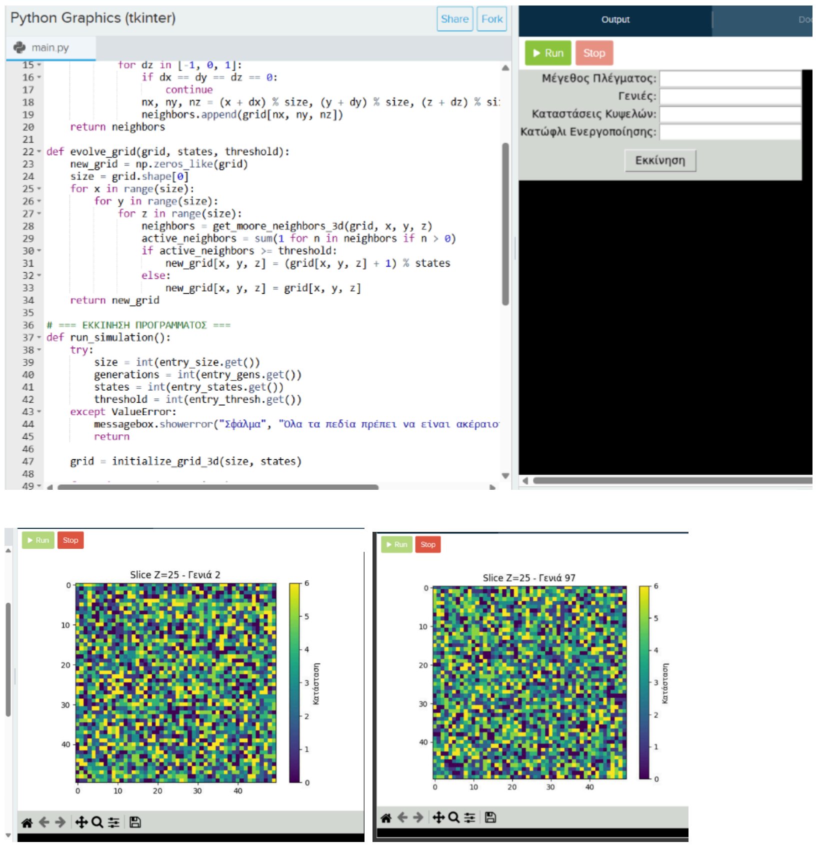
```
import tkinter as tk
from tkinter import messagebox
import numpy as np
import matplotlib.pyplot as plt

# === ΑΡΧΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ ===
def initialize_grid_3d(size, states):
    return np.random.randint(0, states, (size, size, size))

def get_moore_neighbors_3d(grid, x, y, z):
    neighbors = []
    size = grid.shape[0]
    for dx in [-1, 0, 1]:
        for dy in [-1, 0, 1]:
            for dz in [-1, 0, 1]:
                if dx == dy == dz == 0:
                    continue
                nx, ny, nz = (x + dx) % size, (y + dy) % size, (z + dz) % size
                neighbors.append(grid[nx, ny, nz])
    return neighbors

def evolve_grid(grid, states, threshold):
    new_grid = np.zeros_like(grid)
```

```
    size = grid.shape[0]
    for x in range(size):
        for y in range(size):
            for z in range(size):
                neighbors = get_moore_neighbors_3d(grid, x, y, z)
                active_neighbors = sum(1 for n in neighbors if n > 0)
                if active_neighbors >= threshold:
                    new_grid[x, y, z] = (grid[x, y, z] + 1) % states
                else:
                    new_grid[x, y, z] = grid[x, y, z]
    return new_grid

# === ΕΚΚΙΝΗΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ ===
def run_simulation():
    try:
        size = int(entry_size.get())
        generations = int(entry_gens.get())
        states = int(entry_states.get())
        threshold = int(entry_thresh.get())
    except ValueError:
        messagebox.showerror("Σφάλμα", "Όλα τα πεδία πρέπει να είναι ακέραιοι αριθμοί.")
        return

    grid = initialize_grid_3d(size, states)

    for t in range(generations):
        mid_slice = grid[:, :, size // 2]
        plt.imshow(mid_slice, cmap="viridis")
```

```
        plt.title(f"Slice Z={size//2} - Γενιά {t}")
        plt.colorbar(label="Κατάσταση")
        plt.pause(0.3)
        plt.clf()
        grid = evolve_grid(grid, states, threshold)

    plt.show()

# === GUI ===
root = tk.Tk()
root.title("3D Κυψελωτό Αυτόματο")

tk.Label(root, text="Μέγεθος Πλέγματος:").grid(row=0, column=0, sticky="e")
tk.Label(root, text="Γενιές:").grid(row=1, column=0, sticky="e")
tk.Label(root, text="Καταστάσεις Κυψελών:").grid(row=2, column=0, sticky="e")
tk.Label(root, text="Κατώφλι Ενεργοποίησης:").grid(row=3, column=0, sticky="e")

entry_size = tk.Entry(root)
entry_gens = tk.Entry(root)
entry_states = tk.Entry(root)
entry_thresh = tk.Entry(root)

entry_size.grid(row=0, column=1)
entry_gens.grid(row=1, column=1)
entry_states.grid(row=2, column=1)
entry_thresh.grid(row=3, column=1)

tk.Button(root, text="Εκκίνηση", command=run_simulation).grid(row=4, column=0,
columnspan=2, pady=10)
```

root.mainloop()

Χρησιμοποιήθηκε μια περίπτωση από τις 3.



Σχήμα 10:3D-CA με GUI

8.Υλοποίησης με GUI για 2D Κυψελωτό Αυτόματο

```
import tkinter as tk
from tkinter import messagebox
import numpy as np
import matplotlib.pyplot as plt

# === ΓΕΙΤΟΝΙΕΣ ===
def get_neighbors(grid, x, y, neighborhood_type="moore"):
    neighbors = []
    rows, cols = grid.shape
    for dx in [-1, 0, 1]:
        for dy in [-1, 0, 1]:
            if dx == dy == 0:
                continue
            if neighborhood_type == "neumann" and abs(dx) + abs(dy) != 1:
                continue
            nx, ny = (x + dx) % rows, (y + dy) % cols
            neighbors.append(grid[nx, ny])
    return neighbors

# === ΕΞΕΛΙΞΗ ΠΛΕΓΜΑΤΟΣ ===
def evolve_grid(grid, states, threshold, neighborhood_type):
    new_grid = np.copy(grid)
    rows, cols = grid.shape
    for x in range(rows):
        for y in range(cols):
            neighbors = get_neighbors(grid, x, y, neighborhood_type)
            active_neighbors = sum(1 for n in neighbors if n > 0)
            if active_neighbors >= threshold:
                new_grid[x, y] = (grid[x, y] + 1) % states
```

```
            else:
                new_grid[x, y] = grid[x, y]
    return new_grid

# === ΕΚΚΙΝΗΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ ===
def run_simulation():
    try:
        size = int(entry_size.get())
        generations = int(entry_gens.get())
        states = int(entry_states.get())
        threshold = int(entry_thresh.get())
        neighborhood_type = var_neigh.get()
    except ValueError:
        messagebox.showerror("Σφάλμα", "Όλα τα πεδία πρέπει να είναι ακέραιοι αριθμοί.")
        return

    grid = np.random.randint(0, states, (size, size))
    for t in range(generations):
        plt.imshow(grid, cmap="viridis")
        plt.title(f"2D CA - Γενιά {t} - Γειτονιά: {neighborhood_type}")
        plt.colorbar(label="Κατάσταση")
        plt.pause(0.3)
        plt.clf()
        grid = evolve_grid(grid, states, threshold, neighborhood_type)

    plt.show()

# === GUI ===
root = tk.Tk()
root.title("2D Κυψελωτό Αυτόματο")
```

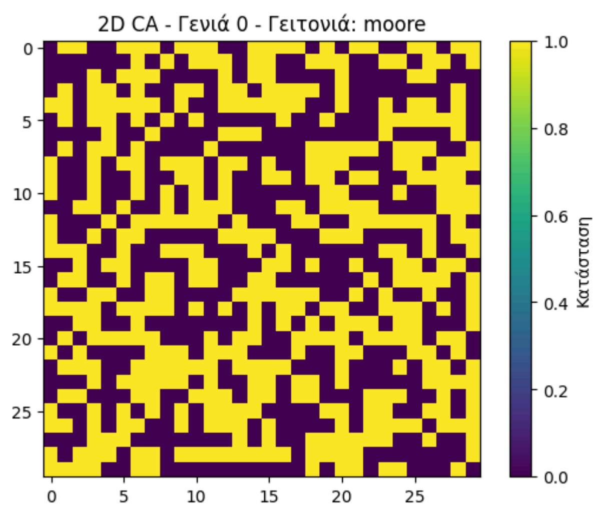
```
tk.Label(root, text="Μέγεθος Πλέγματος:").grid(row=0, column=0, sticky="e")
tk.Label(root, text="Γενιές:").grid(row=1, column=0, sticky="e")
tk.Label(root, text="Καταστάσεις Κυψελών:").grid(row=2, column=0, sticky="e")
tk.Label(root, text="Κατώφλι Ενεργοποίησης:").grid(row=3, column=0, sticky="e")
tk.Label(root, text="Τύπος Γειτονιάς:").grid(row=4, column=0, sticky="e")

entry_size = tk.Entry(root)
entry_gens = tk.Entry(root)
entry_states = tk.Entry(root)
entry_thresh = tk.Entry(root)

entry_size.grid(row=0, column=1)
entry_gens.grid(row=1, column=1)
entry_states.grid(row=2, column=1)
entry_thresh.grid(row=3, column=1)

# Επιλογή Γειτονιάς
var_neigh = tk.StringVar(value="moore")
tk.Radiobutton(root, text="Moore", variable=var_neigh, value="moore").grid(row=4, column=1, sticky="w")
tk.Radiobutton(root, text="Von Neumann", variable=var_neigh, value="neumann").grid(row=4, column=1, sticky="e")
tk.Button(root, text="Εκκίνηση", command=run_simulation).grid(row=5, column=0, columnspan=2, pady=10)
root.mainloop()
```

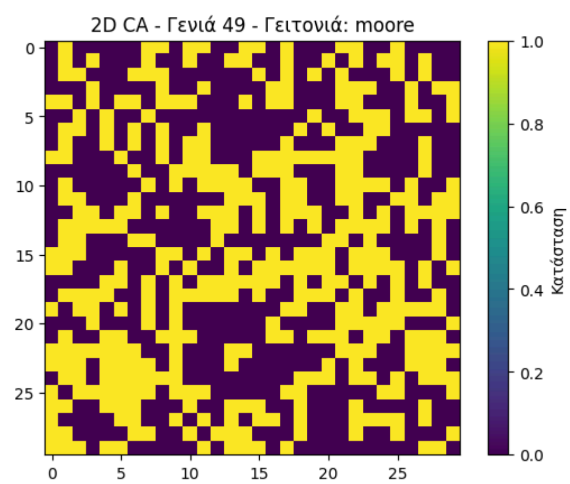
8.2 Υλοποίησης χωρίς GUI για 2D Κυψελωτό Αυτόματο

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# === ΡΥΘΜΙΣΕΙΣ ΧΡΗΣΤΗ ===
size = 30  # πλέγμα 30x30
generations = 50
states = 2
threshold = 3
neighborhood_type = "moore"  # ή "neumann"

# === ΓΕΙΤΟΝΙΕΣ ===
def get_neighbors(grid, x, y, neighborhood_type="moore"):
    neighbors = []
    rows, cols = grid.shape
    for dx in [-1, 0, 1]:
        for dy in [-1, 0, 1]:
            if dx == dy == 0:
                continue
            if neighborhood_type == "neumann" and abs(dx) + abs(dy) != 1:
                continue
            nx, ny = (x + dx) % rows, (y + dy) % cols
            neighbors.append(grid[nx, ny])
    return neighbors

# === ΕΞΕΛΙΞΗ ΠΛΕΓΜΑΤΟΣ ===
def evolve_grid(grid, states, threshold, neighborhood_type):
    new_grid = np.copy(grid)
    rows, cols = grid.shape
    for x in range(rows):
        for y in range(cols):
            neighbors = get_neighbors(grid, x, y, neighborhood_type)
            active_neighbors = sum(1 for n in neighbors if n > 0)
            if active_neighbors >= threshold:
```

```
                new_grid[x, y] = (grid[x, y] + 1) % states
            else:
                new_grid[x, y] = grid[x, y]
    return new_grid

# === ΕΚΚΙΝΗΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ ===
grid = np.random.randint(0, states, (size, size))
for t in range(generations):
    plt.imshow(grid, cmap="viridis")
    plt.title(f"2D CA - Γενιά {t} - Γειτονιά: {neighborhood_type}")
    plt.colorbar(label="Κατάσταση")
    plt.pause(0.3)
    plt.clf()
    grid = evolve_grid(grid, states, threshold, neighborhood_type)

plt.show()
```





9. Συμπεράσματα

Στην παρούσα εργασία μελετήθηκαν τα Κυψελωτά Αυτόματα (Cellular Automata – CA) ως υπολογιστικά μοντέλα για τη διερεύνηση της δυναμικής συστημάτων που εξελίσσονται βάσει κανόνων. Η μελέτη αυτή περιλάμβανε τόσο τη θεωρητική τεκμηρίωση όσο και την προγραμματιστική υλοποίηση σε γλώσσα Python, με στόχο την ανάλυση της συμπεριφοράς διαφορετικών τύπων κυψελωτών αυτομάτων υπό παραμετρικές μεταβολές. Πραγματοποιήθηκαν προσομοιώσεις σε μονοδιάστατο, δισδιάστατο και τρισδιάστατο χώρο, τόσο με όσο και χωρίς χρήση γραφικού περιβάλλοντος (GUI), δίνοντας τη δυνατότητα για πλήρη εποπτεία της εξέλιξης των συστημάτων.

Κατά τη μονοδιάστατη υλοποίηση, αξιοποιήθηκαν οι βασικές αρχές των στοιχειωδών κανόνων (elementary CA), επιτρέποντας την παρατήρηση της συμπεριφοράς του συστήματος από την αρχική κατάσταση μιας μόνο ενεργής κυψέλης. Η εξέλιξη των κυψελίδων παρουσίασε χαρακτηριστικά που κυμαίνονταν από πλήρως προβλέψιμα και περιοδικά μοτίβα έως και έντονα χαοτικές κατανομές, ανάλογα με τον κανόνα που είχε επιλεγεί (π.χ. κανόνες 30 και 110). Η ανάπτυξη του GUI σε 1D περιβάλλον επέτρεψε την ευκολότερη αλληλεπίδραση με το σύστημα και τη δοκιμή διαφορετικών κανόνων και παραμέτρων χωρίς την ανάγκη τροποποίησης του κώδικα.

Η δισδιάστατη προσομοίωση επεκτείνει τη συμπεριφορά του CA σε επίπεδο, επιτρέποντας πλουσιότερες χωρικές δομές και αλληλεπιδράσεις μεταξύ των κυψελίδων. Με την επιλογή γειτονιάς τύπου Moore ή Von Neumann και την εφαρμογή διαφορετικών thresholds, το σύστημα παρουσίασε σύνθετα μοτίβα που σταδιακά οργανώνονταν σε περιοχές σταθερής ή περιοδικής κατάστασης. Το GUI σε 2D περιβάλλον ενίσχυσε σημαντικά την κατανόηση της επίδρασης παραμέτρων όπως ο αριθμός καταστάσεων και το κατώφλι ενεργοποίησης, καθώς η απεικόνιση της εξέλιξης γινόταν σε πραγματικό χρόνο με εύκολη τροποποίηση των αρχικών ρυθμίσεων από τον χρήστη.

Η τρισδιάστατη υλοποίηση αποτέλεσε το πλέον σύνθετο μοντέλο του έργου. Η χρήση πλέγματος τύπου 50×50×50, με γειτονιά Moore 3D (26 γείτονες ανά κυψέλη), οδήγησε σε εξαιρετικά πολυπλοκότερη συμπεριφορά. Η απεικόνιση μέσω της δισδιάστατης τομής του μέσου επιπέδου (slice Z = mid) επέτρεψε την παρακολούθηση της εξέλιξης, αναδεικνύοντας φαινόμενα αστάθειας, αυτοοργάνωσης και τυχαίας διάταξης κυψελίδων. Παρατηρήθηκαν πολύπλοκες και ασύμμετρες δομές, οι οποίες μεταβάλλονταν δυναμικά ανά γενιά, ιδίως όταν το κατώφλι ενεργοποίησης και ο αριθμός καταστάσεων ήταν υψηλοί. Το GUI στο 3D περιβάλλον αποδείχθηκε ιδιαίτερα χρήσιμο, καθώς προσέφερε την αναγκαία ευελιξία στον ορισμό παραμέτρων και τη ζωντανή οπτικοποίηση της εξέλιξης του συστήματος.

Συνολικά, τα πειράματα κατέδειξαν ότι η πολυπλοκότητα του συστήματος δεν είναι απαραίτητα συνάρτηση περίπλοκων κανόνων, αλλά μπορεί να προκύψει και από απλούς τοπικούς κανόνες σε συνδυασμό με χωρική διάσταση και παραμετρικές ρυθμίσεις. Ο αριθμός των καταστάσεων, το κατώφλι ενεργοποίησης, το μέγεθος του πλέγματος και ο τύπος της γειτονιάς αποδείχθηκαν κρίσιμες παράμετροι που καθορίζουν τη συμπεριφορά των CA. Μικρές μεταβολές στις αρχικές συνθήκες ή στους κανόνες οδήγησαν σε θεμελιωδώς διαφορετικά μοτίβα, γεγονός που επιβεβαιώνει τη μη γραμμική φύση αυτών των συστημάτων.

Η εργασία ανέδειξε όχι μόνο τις υπολογιστικές δυνατότητες αυτών των μοντέλων, αλλά και τη σημασία της παραμετροποίησης και της οπτικοποίησης για την πλήρη κατανόηση της εξελικτικής τους πορείας.

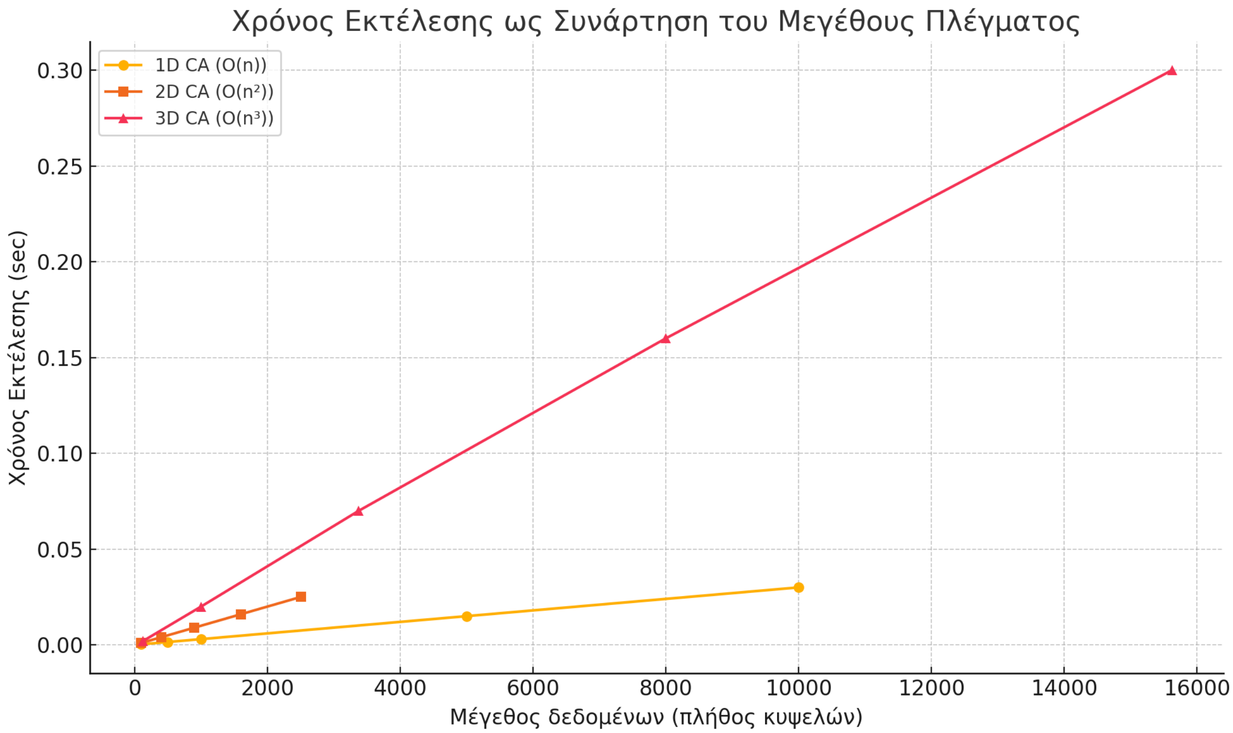
10.Προεκτάσεις και Προτάσεις για Μελλοντική Έρευνα

Δημιουργία Περισσότερων Καταστάσεων και Κατωφλίων: Η προσθήκη ακόμα περισσότερων καταστάσεων για τις κυψελίδες ή η διαφοροποίηση των κατωφλίων ενεργοποίησης μπορεί να αποκαλύψει νέες και πιο περίπλοκες συμπεριφορές. Περαιτέρω διερεύνηση των αλληλεπιδράσεων μεταξύ περισσότερων καταστάσεων θα μπορούσε να οδηγήσει σε πιο εξειδικευμένα και ακριβή μοντέλα.

Εξερεύνηση άλλων Διαστάσεων (4D, 5D): Η προσθήκη επιπλέον διαστάσεων (π.χ. 4D, 5D) μπορεί να αποκαλύψει ακόμα πιο πολύπλοκες και ακατανόητες δυναμικές. Αν και αυτό θα απαιτούσε περισσότερους υπολογιστικούς πόρους, θα μπορούσε να προσφέρει νέες γνώσεις σε τομείς όπως η θεωρία των πολύπλοκων συστημάτων ή η μη γραμμική δυναμική.

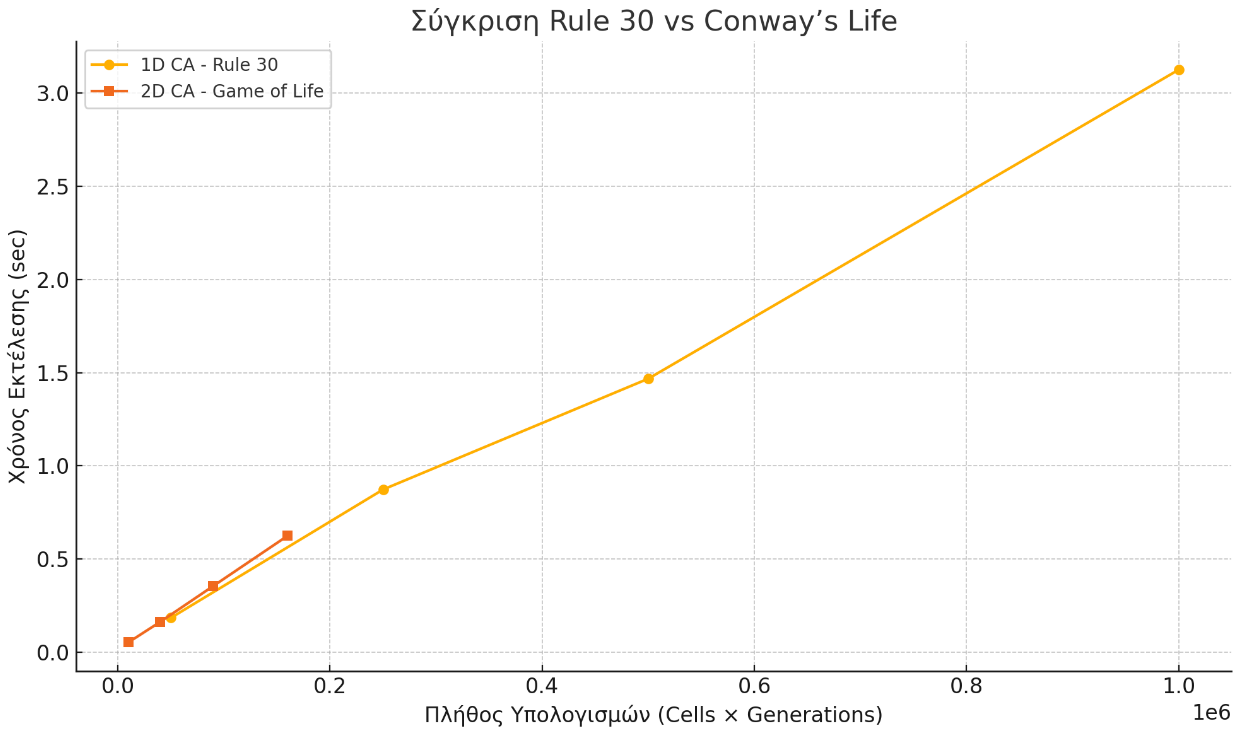
11.Ένθετο

Ανάλυση πολυπλοκότητας



Το διάγραμμα που απεικονίζει τον χρόνο εκτέλεσης σε σχέση με το μέγεθος του πλέγματος για 1D, 2D και 3D Κυψελωτά Αυτόματα. Παρατηρούμε:

Για 1D CA, η αύξηση είναι γραμμική (O(n)).
Για 2D CA, η αύξηση ακολουθεί τετραγωνική πορεία ($O(n^2)$).
Για 3D CA, η αύξηση είναι κυβική ($O(n^3)$) και γίνεται πολύ πιο απαιτητική υπολογιστικά.



Το παραπάνω διάγραμμα συγκρίνει τον χρόνο εκτέλεσης ανάλογα με το μέγεθος του πλέγματος και τον αριθμό γενιών για:

1D Κυψελωτό Αυτόματο με Rule 30

2D Game of Life (Conway's Life)

Συμπεράσματα:

O 1D CA (Rule 30) αυξάνει γραμμικά με το μέγεθος του grid × generations.
O 2D Game of Life αυξάνει πιο απότομα, επιβεβαιώνοντας την τετραγωνική πολυπλοκότητα ($O(n^2)$) λόγω του πλήθους γειτόνων ανά κυψέλη.

12.Βιβλιογραφία

https://www.ceid.upatras.gr/webpages/faculty/papaioan/ICDS2012.pdf

https://sunscrapers.com/blog/data-visualization-python-part1-cellular-automata/

https://github.com/mageirakos/elementary-cellular-automaton

https://pypi.org/project/cellular-automaton/

https://medium.com/@ibrahimmukherjee/cellular-automata-code-in-python-e2eae07b9fe2

https://www.pygame.org/project-Cellular+Automata-559-.html

https://www.fundza.com/algorithmic/form/automata/basic/index.html

https://blog.devgenius.io/simulating-cellular-automata-in-python-f8b74490af69

https://api.arcade.academy/en/2.6.16/example_code/how_to_examples/procedural_caves_cellular.html

https://github.com/DavidColson/CellularAutomata

https://ipython-books.github.io/122-simulating-an-elementary-cellular-automaton/

https://milliams.com/courses/numpy_simulation/Cellular%20automata.html

https://github.com/nicholasyager/cellular-automata

https://www.kaggle.com/code/serkanpeldek/a-new-kind-of-science-cellular-automata

https://www.hashbangcode.com/article/conways-game-life-tkinter-python

https://www.geeksforgeeks.org/cellular-automaton-discrete-model/

https://www.cs.cornell.edu/info/people/tt/Decks/Chapter13.python.pdf

https://faingezicht.com/articles/2017/01/23/wolfram/