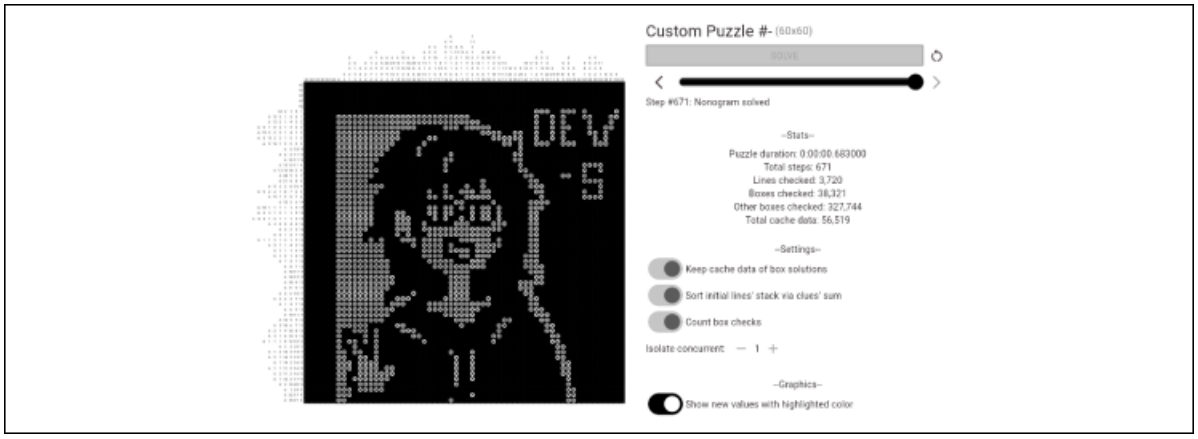
ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Εικονόσταυρο: Δημιουργία Λύτη Παιχνιδιού και Σχεδιαστή Πίστας»



Της φοιτήτριας
Μακρυγιαννάκη Στεφανία Μαρία
Αρ. Μητρώου: 164703

Επιβλέπων
Γουλιάνας Κωνσταντίνος
Βαθμίδα Καθηγητής

26 Ιανουαρίου 2025

Τίτλος Δ.Ε. Εικονόσταυρο: Δημιουργία Λύτη Παιχνιδιού και Σχεδιαστή Πίστας
Κωδικός Δ.Ε. 22310
Ονοματεπώνυμο φοιτήτριας Μακρυγιαννάκη Στεφανία Μαρία
Ονοματεπώνυμο εισηγητή Γουλιάνας Κωνσταντίνος
Ημερομηνία ανάληψης Δ.Ε. 29-10-2022
Ημερομηνία περάτωσης Δ.Ε. 26-01-2025

Βεβαιώνω ότι είμαι η συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

«Αφιερωμένο στη μικρή μου εαυτή και την αδιάκοπη αγάπη της
για τα εικονόσταυρα και τους αλγορίθμους.»

Πρόλογος

Ο λόγος που επέλεξα αυτή την εργασία είναι η βαθιά σύνδεσή μου με τα Εικονόσταυρα, ένα είδος παζλ που λύναμε με τον αδερφό μου από μικρή ηλικία. Αυτή η δραστηριότητα ήταν για μένα κάτι περισσότερο από μια απλή ψυχαγωγία· ήταν μια δημιουργική ασχολία που με χαλάρωνε και με ενθουσίαζε ταυτόχρονα, ενώ παράλληλα ενίσχυε τις λογικές μου δεξιότητες. Τα Εικονόσταυρα αποτελούν σημαντικό κομμάτι της προσωπικής μου πορείας, καθώς συνδυάζουν πρόκληση και ικανοποίηση, ενώ μου έδωσαν τη βάση να αναζητώ συνεχώς νέους τρόπους επίλυσής τους.

Η απόφαση να αναπτύξω έναν αυτόματο λύτη δεν ήταν τυχαία. Ο λύτης αυτός δεν είναι απλώς ένα εργαλείο επίλυσης, αλλά το θεμέλιο για την υλοποίηση εφαρμογών που έχω ήδη σχεδιάσει. Οι ιδέες μου για αυτές τις εφαρμογές εξαρτώνται από τη δυνατότητα ύπαρξης ενός αποδοτικού και λειτουργικού λύτη, ο οποίος θα λειτουργεί παράλληλα, υποστηρίζοντας τη συνολική εμπειρία των χρηστών. Μέσα από την εργασία αυτή, επιδιώκω να συνδυάσω την προσωπική μου αγάπη για τα παζλ με τις γνώσεις μου στην τεχνολογία, δημιουργώντας μια καινοτόμα βάση που θα με βοηθήσει να εξελίξω περαιτέρω τα σχέδιά μου για το μέλλον.

Περίληψη

Η παρούσα διπλωματική εργασία επικεντρώνεται στην ανάπτυξη ενός αυτοματοποιημένου λύτη για το παζλ Εικονόσταυρο, ένα λογικό παζλ όπου οι παίκτες αποκαλύπτουν κρυμμένες εικόνες συμπληρώνοντας κελιά ενός πλέγματος με βάση αριθμητικές ενδείξεις. Κύριος στόχος είναι η δημιουργία μιας εφαρμογής, βασισμένης στο Flutter, η οποία όχι μόνο παρέχει λύσεις, αλλά επεξηγεί τη συλλογιστική πίσω από κάθε βήμα, προσφέροντας τόσο λειτουργικότητα όσο και εκπαιδευτική αξία.

Η έρευνα εξετάζει το θεωρητικό υπόβαθρο των Εικονόσταυρων, αναλύει τις υπολογιστικές προκλήσεις τους ως NP-πλήρη προβλήματα και αξιολογεί υπάρχουσες λύσεις, επισημαίνοντας τις αδυναμίες τους. Προτείνεται και υλοποιείται μια νέα προσέγγιση που συνδυάζει αποδοτικούς αλγορίθμους με φιλική προς τον χρήστη διεπαφή. Οι βασικές τεχνολογίες που χρησιμοποιήθηκαν περιλαμβάνουν το Flutter για την ανάπτυξη εφαρμογών πολλαπλών πλατφορμών και το Firebase για ενδεχόμενη διαχείριση δεδομένων και φιλοξενία.

Η εφαρμογή που αναπτύχθηκε επιτρέπει στους χρήστες να αλληλεπιδρούν με τη διαδικασία επίλυσης, να αναθεωρούν ενδιάμεσα βήματα μέσω ενός συρόμενου ελεγκτή (slider) και να κατανοούν τις λογικές μεθόδους που εφαρμόζει ο λύτης. Τα αποτελέσματα απόδοσης αναδεικνύουν την αποτελεσματικότητα του λύτη και την ικανότητά του να διαχειρίζεται ποικίλης πολυπλοκότητας παζλ. Τα ευρήματα υπογραμμίζουν τη δυνατότητα του Flutter να υποστηρίζει εφαρμογές υψηλών υπολογιστικών απαιτήσεων.

Συνδυάζοντας λειτουργικότητα και εκπαιδευτική διάσταση, η εργασία συμβάλλει τόσο στην κοινότητα επίλυσης παζλ όσο και στην εξερεύνηση προχωρημένων υπολογιστικών μεθόδων σε προσβάσιμες λύσεις λογισμικού.

«Εικονόσταυρο: Δημιουργία Λύτη Παιχνιδιού και Σχεδιαστή Πίστας»

(Nonogram: Development of a Puzzle Solver and Designer)

«Μακρυγιαννάκη Στεφανία Μαρία»

(Makrygiannaki Stefania Maria)

Abstract

This thesis focuses on the development of an automated solver for the Nonogram puzzle, a type of logic puzzle where players uncover hidden images by filling cells in a grid based on numerical clues. The main objective is to create a Flutter-based application that not only provides solutions but also explains the reasoning behind each step, offering both a functional tool and an educational resource.

The research explores the theoretical underpinnings of Nonograms, delves into the computational challenges they pose as NP-complete problems, and evaluates existing solutions, highlighting their limitations. It proposes and implements a novel approach combining efficient algorithms with user-friendly interfaces. Key technologies used include Flutter for cross-platform application development and Firebase for potential data handling and hosting.

The resulting application allows users to interact with the solution process, review intermediate steps via a slider, and understand the logical methods employed by the solver. Performance benchmarks demonstrate the solver's efficiency and its ability to handle various complexities of Nonogram puzzles. The findings underline the potential of Flutter as a framework for high-performance applications in logic and computation-heavy domains.

By bridging functionality with user education, this work contributes to both the puzzle-solving community and the exploration of advanced computing methods in accessible software solutions.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω όλους τους ανθρώπους που με φρόντισαν και με στήριξαν καθ' όλη τη διάρκεια της ακαδημαϊκής μου πορείας. Ευχαριστώ τους γονείς μου, που υπήρξαν το σταθερό μου θεμέλιο και η μεγαλύτερη πηγή στήριξης, την σχέση μου για τη φροντίδα και την αγάπη, καθώς και τους φίλους, τις φίλες και τους συμφοιτητές μου για τη συντροφικότητα και τη διαρκή συμπαράσταση. Τέλος, ευχαριστώ και τους καθηγητές και τους συναδέλφους μου για την πολύτιμη καθοδήγησή τους.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
UI	User Interface
UX	User Experience
PR	Pull Request
Regex	Regular Expression

Κεφάλαιο 1ο: Εισαγωγή

1.1 Εισαγωγή στη Διπλωματική Εργασία

Η παρούσα διπλωματική εργασία εστιάζει στην ανάπτυξη ενός αυτοματοποιημένου λύτη για τα λογικά παζλ ”Εικονόσταυρα”. Τα Εικονόσταυρα είναι λογικά παζλ που απαιτούν την ανάλυση αριθμητικών ενδείξεων για την αποκάλυψη μιας κρυμμένης εικόνας σε ένα πλέγμα. Το παρόν έργο στοχεύει στην υλοποίηση μιας εφαρμογής, βασισμένης στο **Flutter**, η οποία προσφέρει τόσο τη δυνατότητα αυτόματης επίλυσης όσο και εκπαιδευτική αξία, επεξηγώντας τη λογική που εφαρμόζεται σε κάθε βήμα επίλυσης.

Η εργασία ενσωματώνει σύγχρονες τεχνολογίες και μεθοδολογίες για την επίλυση και το σχεδιασμό Εικονόσταυρων. Με την ενσωμάτωση ευρετικών αλγορίθμων και τη χρήση μιας φιλικής προς τον χρήστη διεπαφής, η εφαρμογή συνδυάζει την τεχνολογική καινοτομία με την πρακτική χρησιμότητα.

1.2 Σκοπός της Εργασίας

Ο κύριος σκοπός της διπλωματικής είναι η δημιουργία μιας ολοκληρωμένης εφαρμογής που:

- Εξασφαλίζει την αυτόματη επίλυση Εικονόσταυρων με ακρίβεια και αποτελεσματικότητα.
- Παρέχει εκπαιδευτικό περιεχόμενο, επεξηγώντας τα βήματα επίλυσης.
- Υποστηρίζει τη δημιουργία και την παραμετροποίηση νέων παζλ από τους χρήστες.
- Διασφαλίζει την προσβασιμότητα χωρίς την ανάγκη εγκατάστασης επιπρόσθετου λογισμικού.

Η επιλογή του **Flutter** ως το κύριο πλαίσιο ανάπτυξης αντανακλά την πρόθεση να δημιουργηθεί μια πολυπλατφορμική εφαρμογή που λειτουργεί σε κινητές συσκευές και ιστούς.

1.3 Δομή της Εργασίας

Η εργασία αποτελείται από δεκατρία διακριτά κεφάλαια. Το πρώτο κεφάλαιο, η εισαγωγή, παρουσιάζει τους στόχους, το εύρος και τη σημασία της εργασίας. Περιγράφονται οι στόχοι του έργου, η σύνδεσή του με τις ανάγκες της τεχνολογικής κοινότητας και οι λόγοι επιλογής των συγκεκριμένων εργαλείων.

Το δεύτερο κεφάλαιο εξετάζει το θεωρητικό υπόβαθρο των Εικονόσταυρων. Παρουσιάζονται η ιστορική τους εξέλιξη, οι βασικοί κανόνες και οι ιδιαιτερότητες που τα καθιστούν ενδιαφέροντα λογικά παζλ. Επιπλέον, αναλύονται οι διαφορετικές προσεγγίσεις που έχουν εφαρμοστεί στη λύση τους.

Το τρίτο κεφάλαιο επικεντρώνεται στις τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής. Γίνεται εκτενής αναφορά στο **Flutter** και το **Firebase**, επισημαίνοντας τα πλεονεκτήματά τους και τις προκλήσεις που αντιμετωπίστηκαν κατά τη χρήση τους.

Το τέταρτο κεφάλαιο αναλύει τις μεθοδολογίες επίλυσης που εφαρμόστηκαν στην εργασία. Παρουσιάζονται οι αλγόριθμοι και οι ευρετικές προσεγγίσεις που υλοποιήθηκαν για την επίλυση των παζλ, καθώς και η υπολογιστική πολυπλοκότητα που συνοδεύει τέτοιες εφαρμογές.

Το πέμπτο κεφάλαιο εξετάζει τις επιστημονικές και τεχνολογικές εξελίξεις που σχετίζονται με την επίλυση γρίφων. Αναλύονται υπάρχουσες λύσεις, όπως ο *pbnsolve* και άλλοι διαδικτυακοί λύτες, καθώς και τα σημεία στα οποία υστερούν, θέτοντας τις βάσεις για την ανάπτυξη βελτιωμένων εφαρμογών.

Το έκτο κεφάλαιο επικεντρώνεται στην προετοιμασία της εργασίας. Περιγράφονται οι φάσεις σύλληψης της ιδέας, η συλλογή απαιτήσεων και η οργάνωση των εργασιών, με αναφορά στα εργαλεία που χρησιμοποιήθηκαν, όπως το *GitHub*.

Το έβδομο κεφάλαιο αφορά το σχεδιασμό της εφαρμογής. Παρουσιάζονται οι διαδικασίες δημιουργίας πρωτοτύπων, η χρήση εργαλείων όπως το *Figma*, και η ανάπτυξη διαδραστικών στοιχείων για τη βελτίωση της εμπειρίας χρήστη.

Το όγδοο κεφάλαιο περιγράφει την ανάπτυξη σελίδων, μοντέλων και κύριων γραφικών. Εξηγείται πώς οργανώθηκαν οι λειτουργίες της εφαρμογής, από τη δομή του πλέγματος έως τις διαδραστικές λειτουργίες.

Το ένατο κεφάλαιο επικεντρώνεται στον αυτόματο λύτη, αναλύοντας τους αλγόριθμους και τις τεχνικές που χρησιμοποιήθηκαν για τη δημιουργία του. Περιλαμβάνονται λεπτομέρειες για την επίλυση γραμμών και τη διαχείριση δεδομένων.

Το δέκατο κεφάλαιο αφορά τη διαδικασία σχεδιασμού και προσθήκης παζλ. Περιγράφεται πώς οι χρήστες μπορούν να δημιουργούν ή να τροποποιούν παζλ μέσω της εφαρμογής.

Το ενδέκατο κεφάλαιο εξετάζει τις προκλήσεις και τις λύσεις που προέκυψαν κατά την ανάπτυξη. Γίνεται αναφορά σε ζητήματα απόδοσης, συμβατότητας και πολυνηματικής υλοποίησης.

Το δωδέκατο κεφάλαιο παρουσιάζει τα αποτελέσματα των δοκιμών. Παρουσιάζονται τα δεδομένα απόδοσης του λύτη σε διάφορα παζλ, αξιολογώντας τη λειτουργικότητα και την ακρίβεια της εφαρμογής.

Το δέκατο τρίτο και τελευταίο κεφάλαιο συνοψίζει τα ευρήματα της εργασίας. Παρουσιάζονται οι βασικές συμβολές, οι περιορισμοί, και προτάσεις για μελλοντικές βελτιώσεις, όπως η ενσωμάτωση νέων χαρακτηριστικών και η βελτιστοποίηση αλγορίθμων.

1.4 Συμβολή της Εργασίας

Η παρούσα εργασία συνεισφέρει σημαντικά τόσο στον ακαδημαϊκό όσο και στον πρακτικό τομέα μέσω:

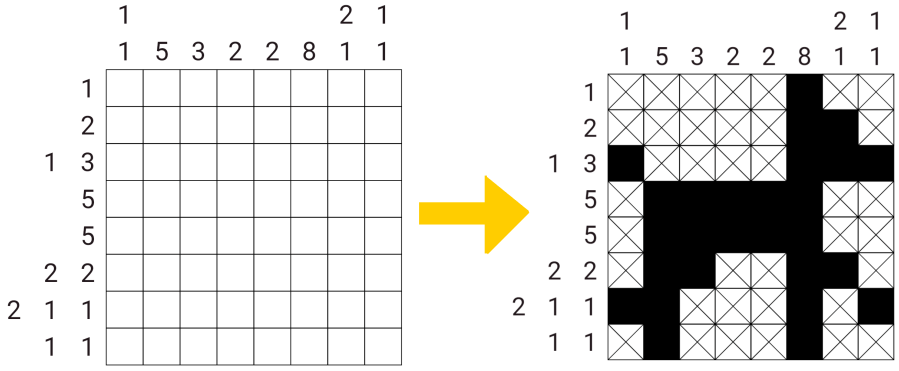
- Της ανάπτυξης ενός πρωτότυπου λύτη στο πλαίσιο του **Flutter**, επιδεικνύοντας τις δυνατότητες του για απαιτητικές εφαρμογές.
- Της εισαγωγής επεξηγηματικών εργαλείων που ενισχύουν τη μαθησιακή εμπειρία των χρηστών.
- Της προώθησης ενός πολυπλατφορμικού εργαλείου, που καθιστά την επίλυση Εικονόσταυρων προσιτή και εύχρηστη σε ευρύτερο κοινό.

Η εργασία αποτελεί σημείο αναφοράς για την περαιτέρω διερεύνηση και ανάπτυξη λύσεων σε παρόμοια προβλήματα λογικής και πολυπλοκότητας.

Κεφάλαιο 2ο: Εικονόσταυρα

2.1 Τι είναι τα Εικονόσταυρα

Τα παζλ Εικονόσταυρα (nonograms), που συναντώνται επίσης ως Νονογράμματα, Χάντζιε (hanjie), Ζωγραφική με Αριθμούς (paint by numbers), Πίκρος (picross), Γκρίντλερς (griddlers) και Πικ-α-Πιξ (pic-a-pix) [1], είναι λογικοί γρίφοι όπου τα κελιά ενός πλέγματος πρέπει να χρωματιστούν ή να παραμείνουν κενά σύμφωνα με τους αριθμούς που δίνονται στις άκρες του πλέγματος, έτσι ώστε να αποκαλυφθεί μια κρυμμένη εικόνα. Συνήθως εμφανίζονται σε ασπρόμαυρη μορφή, αλλά μπορούν να υπάρχουν και σε έγχρωμη εκδοχή.



Σχήμα 2.1: Παράδειγμα εικονόσταυρου με κρυμμένη εικόνα.

2.2 Ιστορία των Εικονόσταυρων

Η ιστορική διαδρομή των εικονόσταυρων ξεκινά από τη Non Ishida (1987), η οποία ανέπτυξε την πρώτη εκδοχή παζλ βασισμένου σε αριθμητικές ενδείξεις για τη συμπλήρωση κελιών. Λίγο αργότερα, ο Tetsuya Nishio (1988) συνέβαλε στη διάδοση αυτής της λογικής με τις «Paint-by-Number» πρακτικές, ενισχύοντας περαιτέρω τη δημοφιλία του νέου είδους παζλ. Σύμφωνα με τον Dalgety (2013) [1], η ονομασία “Nonogram” προέκυψε από το όνομα της Non Ishida και τον χαρακτηρισμό “gram” από την λέγη ”Diagram” (”Διάγραμμα”), υπογραμμίζοντας τον αριθμητικό και διαγραμματικό χαρακτήρα του παζλ. Έκτοτε, τα εικονόσταυρα εξελίχθηκαν σε ένα ξεχωριστό είδος λογικού παζλ, συνδυάζοντας την επεξεργασία απλών αριθμητικών δεδομένων με τη δημιουργία σύνθετων εικόνων, γεγονός που τα καθιστά ιδιαίτερα ελκυστικά τόσο από ψυχαγωγική όσο και από εκπαιδευτική σκοπιά.

2.2.1 Εξώφυλλα Βιβλίων με Εικονόσταυρα

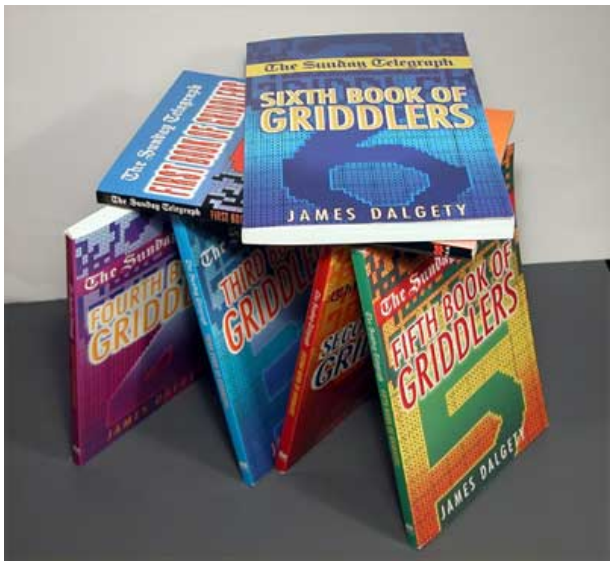
Το 1993 αποτέλεσε κομβικό έτος για τη διάδοσή τους (βλ. [1]), κυρίως χάρη στη Non Ishida και τη δημοσίευσή τους στην εφημερίδα *The Telegraph*. Την ίδια χρονιά, η *Mainichi* στην Ιαπωνία επαναεισήγαγε τα παζλ στο κοινό, ενώ κυκλοφόρησαν και τα πρώτα βιβλία εικονόσταυρων τόσο στην Ιαπωνία όσο και στο Ηνωμένο Βασίλειο. Έκτοτε, εκδόσεις εμφανίστηκαν σε χώρες όπως η Σουηδία, οι Ηνωμένες Πολιτείες και η Νότια Αφρική, εδραιώνοντας την παγκόσμια διάδοσή τους.

Σχήμα 2.2: Παραδείγματα εξωφύλλων βιβλίων με εικονόσταυρα. [1]

2.2.2 Συλλογές “Griddlers”

Μέχρι το 1995 (βλ. [1]), οι εκδόσεις *Pan Books* είχαν ήδη φτάσει στην 4η έκδοση του *Sunday Telegraph Book of Nonograms*. Η Non Ishida επιθυμούσε να διατηρήσει την ονομασία «Nonogram» αποκλειστικά για τα σχέδιά της στην Ιαπωνία, γεγονός που οδήγησε στη λήξη της συνεργασίας της με τον James Dalgety το 1996. Το καλοκαίρι του 1998, η *Sunday Telegraph* οργάνωσε διαγωνισμό για την επιλογή ενός νέου ονόματος για το παζλ, με τους αναγνώστες να καταλήγουν στο «Griddler».

Το φθινόπωρο του 1999, η ίδια εφημερίδα δημοσίευσε το πρώτο *Griddler Book* και από τότε εκδίδει ένα κάθε χρόνο, επιβεβαιώνοντας τη διαρκή δημοφιλία του συγκεκριμένου οπτικολογικού παζλ.



Σχήμα 2.3: Μια στοίβα βιβλίων “Griddlers”. [1]

2.2.3 Ηλεκτρονικά παζλ

Τα παζλ Paint by Numbers χρησιμοποιήθηκαν πολύ σε παιχνίδια χειρός, π.χ., Game Boy, μέχρι και το 1995. Η Nintendo, βλέποντας την απήχηση που είχαν από τον κόσμο, συνέχισε κυκλοφορώντας τα Picross (σταυρόλεξα εικόνας), τα οποία ξεκίνησαν με το Game Boy και έφτασαν μέχρι και το Nintendo

3DS και Switch, όπου κυκλοφόρησαν 9 τίτλοι παιχνιδιών.

Το 2011, ένας Βρετανός προγραμματιστής παιχνιδιών, ο Jagex, λάνσαρε ένα παζλ μη γραμμάτων. Το 2013, η Casual Labs κυκλοφόρησε το Paint it Back, και το 2017 το Picross διαδέχθηκε το Pictopix, το οποίο απευθυνόταν σε υπολογιστές και έδινε τη δυνατότητα στους παίκτες να μοιράζονται τις δημιουργίες τους.

Σήμερα, τα παζλ συνεχίζουν να έχουν αυξημένη δημοτικότητα. Πέρα από τα ηλεκτρονικά παζλ, πολλά περιοδικά σε όλο τον κόσμο περιλαμβάνουν παζλ μη γραμμάτων, καθώς αποτελούν μια διασκέδαση για μικρούς και μεγάλους, ενώ ταυτόχρονα συμβάλλουν στην όξυνση του εγκεφάλου.

Σχήμα 2.4: Το παιχνίδι Picross, στην κονσόλα Nitendo 3DS.

2.3 Κανόνες και λύση εικονόσταυρου

Οι κανόνες του εικονόσταυρου είναι απλοί αλλά απαιτούν λογική σκέψη και στρατηγική. Το παζλ αποτελείται από ένα πλέγμα με κελιά, τα οποία μπορούν να γεμίσουν (συνήθως με μαύρο χρώμα) ή να σημαδευτούν με ένα X για να δηλώσουν ότι είναι κενά. Δίπλα σε κάθε σειρά και στήλη του πλέγματος υπάρχουν αριθμοί που αποτελούν τα στοιχεία επίλυσης. Αυτοί οι αριθμοί υποδεικνύουν πόσα μαύρα διαδοχικά κελιά υπάρχουν, ενώ κάθε ομάδα μαύρων κελιών πρέπει να διαχωρίζεται από τουλάχιστον ένα κενό κελί (σημειωμένο με X).

Η λύση του εικονόσταυρου βασίζεται σε μια συστηματική διαδικασία που συνδυάζει λογική και υπομονή. Οι παίκτες ξεκινούν από τις γραμμές ή στήλες που έχουν τις πιο ”εύκολες” ενδείξεις, δηλαδή εκείνες όπου ο αριθμός των κελιών είναι αρκετός για να καθοριστούν με ακρίβεια οι πρώτες θέσεις. Για παράδειγμα, σε μια σειρά με πέντε κελιά και μία ένδειξη [5], όλα τα κελιά της σειράς γεμίζουν, καθώς δεν υπάρχει περιθώριο για κενά. Αντίστοιχα, σε μεγαλύτερες γραμμές ή στήλες, τα κεντρικά κελιά μπορεί να θεωρηθούν σίγουρα γεμάτα όταν οι ενδείξεις υπαγορεύουν συγκεκριμένες θέσεις.

Μια βασική τεχνική επίλυσης περιλαμβάνει τη διασταύρωση πληροφοριών μεταξύ γραμμών και στη-

λών. Όταν γεμίζεται ένα κελί σε μια γραμμή, αυτό επηρεάζει τις πιθανές λύσεις της αντίστοιχης στήλης, και αντιστρόφως. Με αυτόν τον τρόπο, η λύση εξελίσσεται βήμα-βήμα, καθώς κάθε νέα πληροφορία περιορίζει περαιτέρω τις επιλογές στις υπόλοιπες γραμμές και στήλες.

Ένα άλλο σημαντικό εργαλείο είναι η χρήση της ”λογικής εξάντλησης” (logic exhaustion). Σε αυτή τη μέθοδο, οι παίκτες εξετάζουν όλα τα πιθανά σενάρια για μια γραμμή ή στήλη και καταλήγουν σε ποια κελιά είναι βέβαιο ότι γεμίζουν ή παραμένουν κενά. Αυτή η διαδικασία είναι ιδιαίτερα χρήσιμη σε πιο σύνθετα παζλ, όπου η λύση δεν είναι προφανής από τις πρώτες ενδείξεις.

Στην πράξη, οι έμπειροι λύτες χρησιμοποιούν συχνά διαφορετικούς χρωματισμούς ή σύμβολα για να παρακολουθούν την πρόοδο τους. Για παράδειγμα, μπορεί να χρησιμοποιήσουν διαφορετικά χρώματα για τα σίγουρα γεμάτα κελιά και τα πιθανά, ενώ διαχωρίζουν τα κενά σημεία με διαφορετικά σχήματα.

Η διαδικασία λύσης μπορεί να απαιτεί αρκετές δοκιμές και διορθώσεις, ειδικά σε μεγάλα και πολύπλοκα παζλ. Σε αυτές τις περιπτώσεις, οι λύτες δοκιμάζουν διαφορετικά μοτίβα, σημειώνουν προσωρινές επιλογές και επανεξετάζουν τα δεδομένα όταν μια αρχική υπόθεση αποδεικνύεται εσφαλμένη.

Η επίλυση ενός εικονόσταυρου είναι ένας συνδυασμός λογικής, στρατηγικής και υπομονής. Καθώς το πλέγμα συμπληρώνεται, η εικόνα που προκύπτει γίνεται σταδιακά εμφανής, προσφέροντας μια αίσθηση ικανοποίησης και ανταμοιβής στον παίκτη.

2.4 Τι είναι ο λύτης

Ο λύτης είναι ένα λογισμικό πρόγραμμα ή αλγόριθμος που σχεδιάστηκε για να βρίσκει λύσεις σε συγκεκριμένα προβλήματα ή παζλ. Στο πλαίσιο των παζλ όπως τα Εικονόσταυρα, ο λύτης αυτόματα δημιουργεί αυτοματοποιημένες λύσεις στο παζλ βάσει των δοθέντων περιορισμών και κανόνων. Χρησιμοποιεί διάφορες τεχνικές, όπως η αναδρομή, η διασπορά περιορισμών και η αναγνώριση προτύπων, για να προσδιορίσει συστηματικά τη σωστή διάταξη των κελιών στο πλέγμα του παζλ.

Στη συγκεκριμένη πτυχιακή εργασία, έχουν δημιουργηθεί ορισμένα Εικονόσταυρα στα οποία παρέχεται και ο αντίστοιχος λύτης τους. Επιλέγοντας τη λύση, εμφανίζεται μια μπάρα με την οποία ο παίκτης μπορεί να δει ένα μέρος, ένα σημείο ή και ολόκληρη τη λύση του παζλ. Αυτό αποσκοπεί στο να βοηθήσει τον παίκτη που έχει “κολλήσει” σε ένα σημείο του παζλ προκειμένου να τον βοηθήσει στην επίλυσή του, ή και απλούστερα για να δει το αποτέλεσμα της κρυμμένης εικόνας με μια απλή κίνηση.

Κεφάλαιο 3ο: Τεχνολογίες που χρησιμοποιήθηκαν

3.1 Firebase

Η Firebase είναι μια ολοκληρωμένη πλατφόρμα που αναπτύχθηκε από την Google και προσφέρει μια ποικιλία υπηρεσιών και εργαλείων που έχουν σκοπό να βοηθήσουν στην κατασκευή, βελτίωση και ανάπτυξη εφαρμογών. Είναι ιδιαίτερα δημοφιλής στις εφαρμογές για κινητά τηλέφωνα αλλά και για εφαρμογές περιηγητή ιστού.

Η Firebase δημιουργήθηκε το 2011, αλλά άρχισε να ανήκει στη Google από το 2014. Η πλατφόρμα επεκτάθηκε και ενοποιήθηκε με πολλές υπηρεσίες της Google, όπως το Google Ads και το Google Cloud Platform, προσφέροντας περισσότερα εργαλεία στους προγραμματιστές.

3.1.1 Πλεονεκτήματα της Firebase

Τα πλεονεκτήματα της Firebase περιλαμβάνουν:

- Διαθέτει μια βάση δεδομένων πραγματικού χρόνου.
- Παρέχει μεγάλο χώρο αποθήκευσης.
- Δεν απαιτεί υποδομές και server για να τεθεί σε λειτουργία.
- Διαθέτει ισχυρό σύστημα ασφαλείας.
- Απαιτεί ελάχιστες παραμετροποιήσεις.
- Παρέχει εύκολη ανάκτηση δεδομένων.

Η Firebase περιλαμβάνει μια βάση δεδομένων τύπου NoSQL, η οποία βασίζεται σε τεχνολογίες cloud. Τα δεδομένα οργανώνονται και αποθηκεύονται σε μορφή JSON, επιτρέποντας την αποθήκευση και συγχρονισμό δεδομένων σε πραγματικό χρόνο.

3.1.2 Μειονεκτήματα της Firebase

Τα βασικά μειονεκτήματα της Firebase είναι τα εξής:

- Δεν έχει δοκιμαστεί ευρέως σε εμπορικές εφαρμογές.
- Οι δυνατότητες αναζήτησης είναι περιορισμένες.
- Δεν υποστηρίζει παραγωγή και καταμέτρηση συνόλων.

3.1.3 Χρήση της Firebase στην εργασία

Στην παρούσα εργασία, η Firebase χρησιμοποιείται αποκλειστικά για τη δυνατότητα φιλοξενίας (hosting) της ιστοσελίδας που επιτρέπει τη χρήση του λύτη. Υπάρχει προοπτική για μελλοντική χρήση της βάσης δεδομένων, με στόχο τη βελτίωση της αποθήκευσης δεδομένων και της διατήρησης της κατάστασης του λύτη.

3.2 Το Flutter

Το Flutter είναι ένα κιτ ανάπτυξης λογισμικού της Google (Google UI Kit), το οποίο χρησιμοποιείται για τη δημιουργία εγγενώς μεταγλωττισμένων εφαρμογών για κινητά, ιστότοπους και υπολογιστές από μια ενιαία βάση κώδικα. Στο Flutter χρησιμοποιείται η γλώσσα προγραμματισμού Dart, η οποία είναι βελτιστοποιημένη για γρήγορους κύκλους ανάπτυξης και παρέχει μια πλούσια συλλογή στοιχείων για τη δημιουργία πολύπλοκων γραφικών διεπαφών.

3.2.1 Πλεονεκτήματα του Flutter

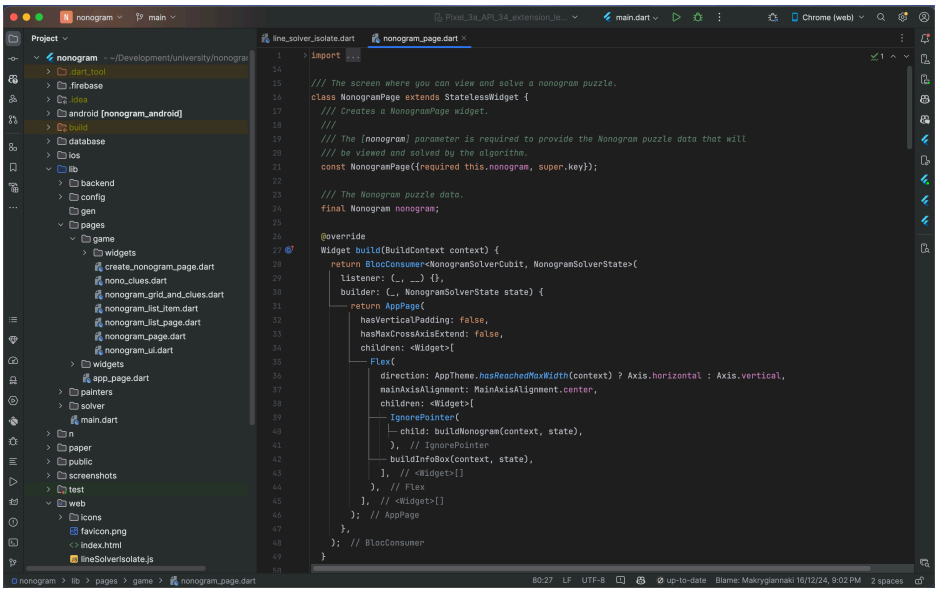
Τα πλεονεκτήματα του Flutter περιλαμβάνουν:

1. **Ένας κώδικας για πολλές πλατφόρμες:** Ο ίδιος κώδικας μπορεί να χρησιμοποιηθεί για τη δημιουργία εφαρμογών σε διαφορετικές πλατφόρμες (κινητά, ιστός, υπολογιστές).
2. **Πλούσιο περιβάλλον χρήστη:** Το Flutter παρέχει προσχεδιασμένα γραφικά, καθώς και τη δυνατότητα δημιουργίας πολύπλοκων στοιχείων διεπαφής.
3. **Γλώσσα Dart:** Η Dart προσφέρει ένα ασφαλές περιβάλλον προγραμματισμού (type-safe), μειώνοντας τα σφάλματα κατά τη μεταγλώττιση.
4. **Εκτέλεση:** Το Flutter διαθέτει μηχανή υψηλής απόδοσης, με ομαλή λειτουργία και άμεσες αλλαγές στον κώδικα.

3.2.2 Μειονεκτήματα του Flutter

Τα μειονεκτήματα του Flutter περιλαμβάνουν:

1. Οι εφαρμογές Flutter απαιτούν περισσότερο αποθηκευτικό χώρο λόγω των βιβλιοθηκών που χρησιμοποιούνται.
2. Η υποστήριξη σε ιστό και υπολογιστές είναι περιορισμένη σε σύγκριση με κινητές συσκευές.
3. Η ποικιλία των βιβλιοθηκών είναι μικρότερη σε σχέση με άλλες τεχνολογίες.
4. Οι εφαρμογές Flutter web έχουν μειωμένη απόδοση σε ορισμένες περιπτώσεις (π.χ., αργή φόρτωση, περιορισμένη SEO).



Σχήμα 3.1: Κώδικας σε Flutter

3.2.3 Χρήση του Flutter στην εργασία

Στην παρούσα εργασία, το Flutter χρησιμοποιήθηκε για την ανάπτυξη της εφαρμογής λύσης εικονόσταυρων. Η επιλογή του Flutter έγινε λόγω της ευελιξίας του και της δυνατότητάς του να αναπτύσσει εφαρμογές για πολλαπλές πλατφόρμες με κοινή βάση κώδικα.

3.3 Πεδίο εφαρμογής και περιορισμοί

Η εργασία επικεντρώνεται στην ανάπτυξη μιας εφαρμογής επίλυσης Nonogram που, εκτός από την παροχή λύσεων, προσφέρει και λεπτομερείς εξηγήσεις για τα βήματα της διαδικασίας επίλυσης. Ωστόσο, λόγω των υπολογιστικών περιορισμών του Flutter, η εφαρμογή δεν μπορεί να διαχειριστεί πολύ μεγάλα ή σύνθετα παζλ με απόδοση ανάλογη πιο εξειδικευμένων λύσεων χαμηλού επιπέδου (π.χ., σε C ή Assembly).

Κεφάλαιο 4ο: Ανάλυση μεθοδολογιών επίλυσης

4.1 Εισαγωγή

Στο παρόν κεφάλαιο γίνεται ανάλυση της βιβλιογραφίας και επιστημονικών ερευνών που έχουν πραγματοποιηθεί σχετικά με την επίλυση εικονόσταυρων. Η ανάλυση αυτή θα βοηθήσει στην καλύτερη κατανόηση της παρούσας έρευνας.

Κατά την ανάλυση, εξετάζονται ποικίλες προσεγγίσεις και μέθοδοι που έχουν χρησιμοποιηθεί για την επίλυση εικονόσταυρων, καθώς και τεχνικές που έχουν αναπτυχθεί για την αυτοματοποιημένη επίλυση τους. Δίνεται έμφαση στα επιστημονικά και τεχνολογικά επιτεύγματα τα οποία έχουν χρησιμοποιούνται μέχρι σήμερα και αναλύονται τυχόν κενά ή περιορισμοί στις υπάρχουσες λύσεις. Τέλος, βάση των αναγκών και προκλήσεων που εντοπίστηκαν κατά την έρευνα ακολουθούν τα δικά μου ευρήματα και συμπεράσματα για την επίλυση εικονόσταυρων.

4.2 Μεθοδολογίες Επίλυσης Εικονοσταύρων

Οι μεθοδολογίες που επικεντρώνονται στην επίλυση των εικονόσταυρων δεν πρόκειται για αλγορίθμους που έχουν χρησιμοποιηθεί απαραίτητα σε λύτες, αλλά για αλγόριθμους και μεθόδους που προσφέρονται για οποιονδήποτε επιθυμεί να επιλύσει μόνος ένα εικονόσταυρο.

Εξετάζονται διάφορες μέθοδοι που έχουν προταθεί σε ιστοσελίδες με παζλ εικονοσταύρων, ακαδημαϊκές έρευνες και δημοσιευμένες εργασίες, με έμφαση στην εμπειρία του χρήστη κατά την επίλυση τους. Εξετάζεται πώς οι αλγόριθμοι και οι μέθοδοι αυτοί προσφέρουν ένα φιλικό και εύχρηστο περιβάλλον για την επίλυση των εικονόσταυρων.

Στόχος αυτής της ανασκόπησης είναι να διαπιστώσουμε πώς οι μεθοδολογίες αυτές μπορούν να επιτρέψουν στους χρήστες να επιλύουν τα εικονόσταυρα με μεγαλύτερη άνεση και επιτυχία, και τους παράγοντες που επηρεάζουν την αποτελεσματικότητά τους. Αυτή η κατανόηση είναι απαραίτητη ως προς τη διαμόρφωση μιας πιο χρήσιμης και φιλικής για τον χρήστη εφαρμογής.

Παρακάτω αναλύονται οι μεθοδολογίες επίλυσης σε: επίλυση ανά γραμμή, επίλυση ανά πολλαπλές γραμμές, επίλυση με εκτιμήσεις και τέλος επίλυση με μαθηματικές πράξεις.

4.2.1 Επίλυση ανά γραμμή

4.2.1.1 Εισαγωγή

Η πιο βασική και δημοφιλής μέθοδος επίλυσης είναι η μελέτη κάθε γραμμής μόνη της. Αυτή η προσέγγιση θεωρείται ευρέως προσβάσιμη και δημοφιλής λόγω της απλότητάς της και της ικανότητάς της να επιλύει το μεγαλύτερο ποσοστό των παζλ.

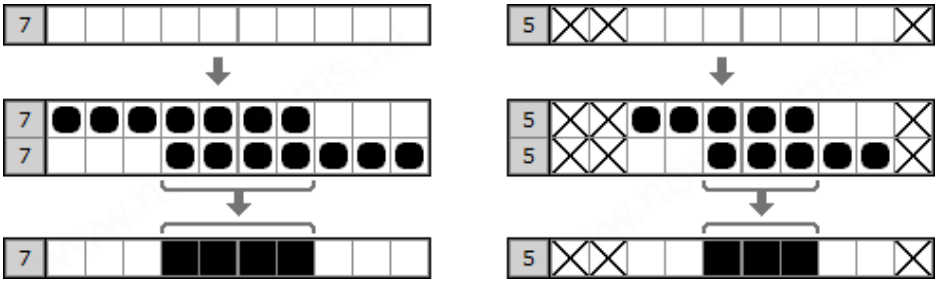
Παρόλο που η δημοφιλέστερη προσέγγιση αποδεικνύεται αποτελεσματική στην επίλυση πολλών παζλ, είναι εμφανές ότι δυσκολεύεται να ανταποκριθεί σε παζλ με περίπλοκη δομή ή υψηλότερο επίπεδο δυσκολίας. Έτσι, παρότι απλή στην εφαρμογή της, η αρχική προσέγγιση μπορεί να φτάσει σε σημείο όπου η

επίλυση του παζλ απαιτεί την χρήση εξειδικευμένων τεχνικών ή μεθόδων, όπως η ανάλυση συνδυασμών γραμμών ή η πρόβλεψη λύσεων. [3] [2]

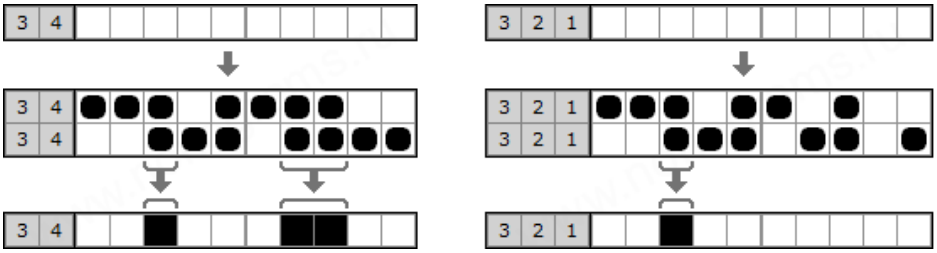
4.2.1.2 Επικάλυψη

Η πρώτη και πιο συνηθισμένη προσέγγιση επίλυσης ανά γραμμή αφορά την επικάλυψη στοιχείων των λύσεων. Η λογική αυτής της λύσης αφορά την καταγραφή των λύσεων τμηματικά. Έτσι, σε οποιαδήποτε λύση, το αποτέλεσμα του κουτιού είναι το ίδιο.

Η τακτική που ακολουθείται για την εύρεση αυτών των λύσεων, είναι να γίνεται σημείωση δύο πιθανών λύσεων: της πιο αριστερής και της πιο δεξιάς. Αυτές τις λύσεις τις βρίσκουμε αν ξεκινήσουμε να τοποθετούμε τα στοιχεία αντικριστά πάνω στα πλαίσια του πλέγματος μια από τα αριστερά και μια από τα δεξιά. Όπου συμπίπτουν στοιχεία ή κενά, τα συμπληρώνουμε καθώς είναι η μοναδική λύση αυτού του πλαισίου. (Σχήμα 4.1 και Σχήμα 4.2)



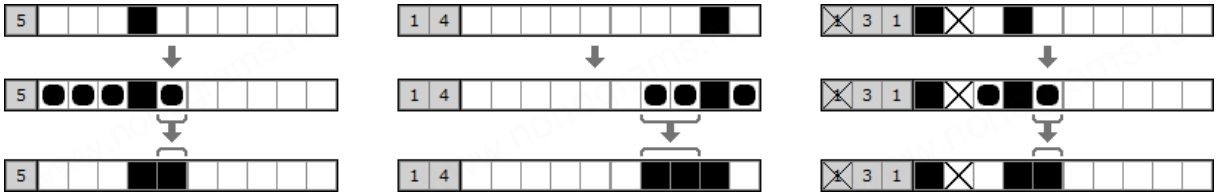
Σχήμα 4.1: Πρώτο παράδειγμα ”Επικάλυψης”



Σχήμα 4.2: Δεύτερο παράδειγμα ”Επικάλυψης”. [2]

4.2.1.3 Σπρώχνοντας από τους τοίχους

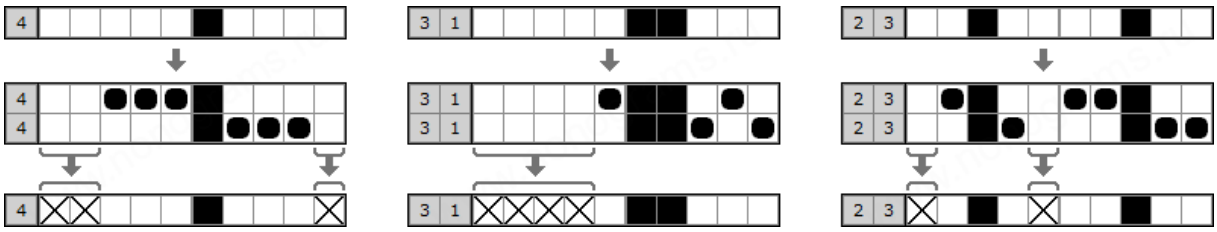
Σε μια πιο προηγμένη φάση, όταν έχουν ήδη τοποθετηθεί ορισμένα κομμάτια της λύσης, μπορεί να εφαρμοστεί η μέθοδος του ”σπρωξίματος” από τους τοίχους. Αρχίζοντας από το αριστερότερο ή δεξιότερο σημείο όπου μπορεί να τοποθετηθεί ένα στοιχείο, ελέγχουμε αν υπερβαίνει το ήδη τοποθετημένο στοιχείο, και εάν ναι, το συμπληρώνουμε. (Σχήμα 4.3)



Σχήμα 4.3: Παράδειγμα “Σπρώχνοντας από τους τοίχους”. [2]

4.2.1.4 Αδυνατότητα πρόσβασης

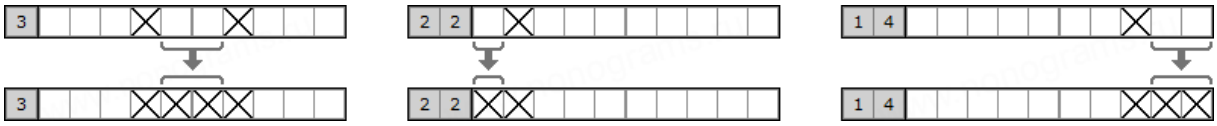
Στην τεχνική αδυνατότητας πρόσβασης, προχωρούμε στον αποκλεισμό των κελιών στα οποία δεν υπάρχει δυνατότητα να εισαχθεί οποιαδήποτε πιθανή λύση, με βάση τα υπάρχοντα στοιχεία του παζλ. (Σχήμα 4.4)



Σχήμα 4.4: Παράδειγμα Αδυνατότητας Πρόσβασης. [2]

4.2.1.5 Δεν χωράει

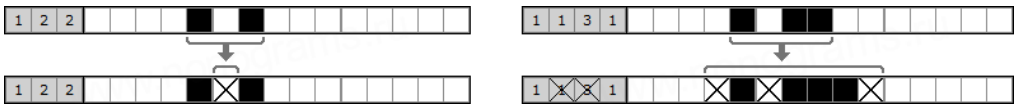
Συνεχίζοντας με μια ανάλογη λογική προσέγγιση, προχωρούμε στον αποκλεισμό κελιών που βρίσκονται σε θέσεις που δεν μπορούν να φιλοξενήσουν στοιχεία σε αριθμό ίσο ή μικρότερο από τα εναπομείναντα για συμπλήρωση. (Σχήμα 4.5)



Σχήμα 4.5: Παράδειγμα “Δεν χωράει”. [2]

4.2.1.6 Διαίρεση

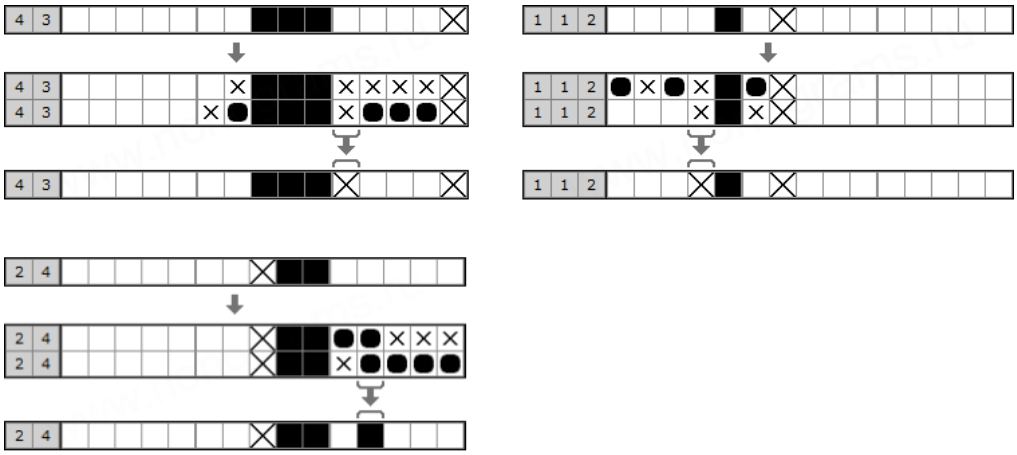
Στην περίπτωση της Διαίρεσης, ακολουθούμε μια διαδικασία διαχωρισμού των ήδη υπαρκτών κελιών. Όταν ο συνολικός αριθμός τους, συν το διάστημα που τα χωρίζει, υπερβαίνει το μεγαλύτερο σε αριθμό στοιχείο από το σύνολο όλων των διαθέσιμων στοιχείων προς τοποθέτηση, τότε το ενδιάμεσο κελί είναι κενό. (Σχήμα 4.6)



Σχήμα 4.6: Παράδειγμα Διαίρεσης. [2]

4.2.1.7 Διπλή τοποθέτηση

Ακολουθώντας την ίδια λογική, η μέθοδος της Διπλής Τοποθέτησης αποσκοπεί επίσης στο να αποκλείσει κουτιά. Για να εντοπίσουμε αυτά τα κουτιά, ξεκινάμε επιλέγοντας ήδη συμπληρωμένα κουτιά που ανήκουν σε περισσότερα από ένα διαθέσιμα στοιχεία. Στη συνέχεια, αξιολογούμε όλες τις πιθανές τοποθετήσεις για κάθε ένα από αυτά τα στοιχεία. Αν υπάρχουν κουτιά που δεν μπορούν να συμπληρωθούν με καμία από τις πιθανές λύσεις, τότε αυτά μπορούν να αποκλειστούν. (Σχήμα 4.7)



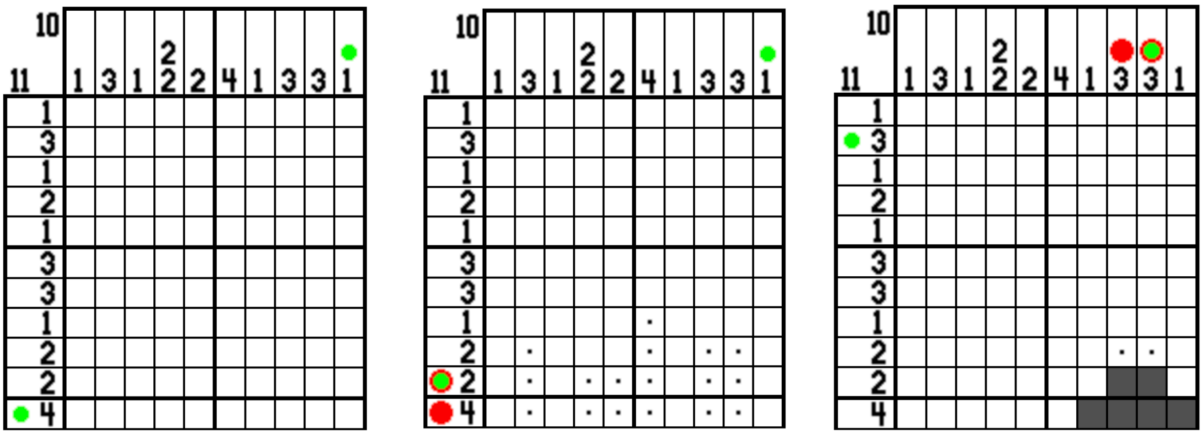
Σχήμα 4.7: Παράδειγμα Διπλής Τοποθέτησης. [2]

4.2.2 Επίλυση με πολλαπλές γραμμές

Η επίλυση με πολλαπλές γραμμές είναι πιο περίπλοκη από αυτή της ανά γραμμής αλλά είναι πιο αποδοτική. Καθώς προσθέτοντας τη δεύτερη διάσταση στον τρόπο επίλυσης του εικονόσταυρου γίνεται να χρησιμοποιηθούν δύο μέθοδοι ανά γραμμή ταυτόχρονα. Έτσι μπορεί να θεωρηθεί πιο δύσκολη μεθοδολογία αλλά μπορεί να επιλύσει πιο περίπλοκα και δυσκολότερα εικονόσταυρα.

4.2.2.1 Μέθοδος Ακμής

Το πλεονέκτημα της συμπλήρωσης κουτιών στις άκρες του πλέγματος είναι ότι αν είναι γνωστά τα στοιχεία στις κάθετες γραμμές ή στήλες, μπορούν να συμπληρωθούν ολόκληρες γραμμές ή στήλες. Αυτή την πληροφορία εκμεταλλεύεται η μέθοδος της ακμής, καθώς εξετάζει τη συμπλήρωση μιας γραμμής άκρης, της προηγούμενής της, και το αντίστοιχο ακραίο στοιχείο από τις κάθετες σε αυτές γραμμές ή στήλες. (Σχήμα 4.8)

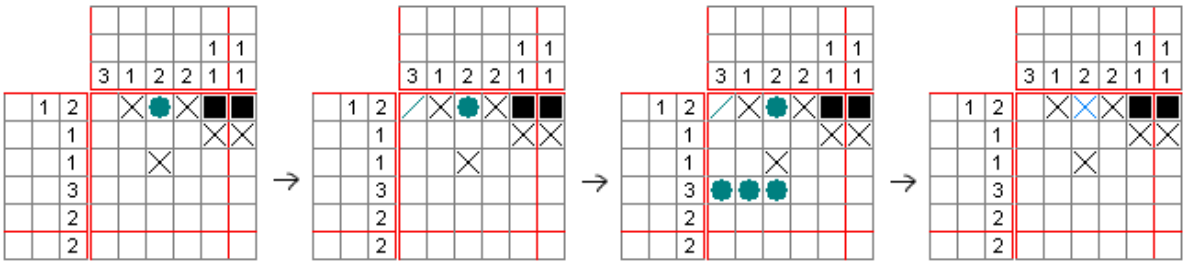


Σχήμα 4.8: Παράδειγμα Μεθόδου Ακμής. [3]

4.2.2.2 Μέθοδος Υπόθεσης

Ως τελευταία μέθοδος επίλυσης με λογική που μπορεί να ακολουθηθεί, αναφέρεται η τεχνική της Υπόθεσης. Σύμφωνα με αυτήν την προσέγγιση, η επίλυση του παζλ ξεκινάει κρατώντας εγγραμμένες τις κινή-

σεις που έχουμε κάνει με διαφορετικό τρόπο, προσπαθώντας να παρατηρήσουμε οποιοδήποτε σφάλμα. Μόλις αντικρίσουμε κάποιο, επιστρέφουμε στην αρχική υπόθεσή μας και τοποθετούμε την αντίθετη λύση από αυτήν που είχαμε αρχικά προβλέψει. (Σχήμα 4.9)



Σχήμα 4.9: Παράδειγμα Μεθόδου Υπόθεσης

4.2.3 Επίλυση με μαθηματικές πράξεις

4.2.3.1 Κωδικοποίηση μεθοδολογίας Επικάλυψης

Μια μαθηματική προσέγγιση ασχολείται με την κωδικοποίηση της πρώτης μεθοδολογίας που αναφέρθηκε σε αυτό το κεφάλαιο, την ”Επικάλυψη”. Για να γίνει αυτό, πρώτο βήμα είναι η άθροιση των στοιχείων της γραμμής ή στήλης, συμπεριλαμβανομένων των κενών μεταξύ τους. Λαμβάνεται ο υπολειπόμενος διαθέσιμος χώρος αφαιρώντας το άθροισμα από το συνολικό πλάτος ή ύψος του παζλ.

Η τεχνική αυτή υποδεικνύει πόσα κουτάκια μπορούν να συμπληρωθούν για κάθε στοιχείο. Αν το στοιχείο υποδεικνύει έναν αριθμό μεγαλύτερο από το αποτέλεσμα της αφαίρεσης, τότε μπορούν να συμπληρωθούν κάποια από τα κουτάκια του.

Για παράδειγμα, αν το άθροισμα των στοιχείων σε μια γραμμή είναι 20 και το παζλ έχει πλάτος 23, η αφαίρεση είναι 3. Αν αυτή η γραμμή περιέχει ένα στοιχείο αριθμού 5, αυτό σημαίνει πως 2 κουτάκια αυτού του 5 μπορούν να συμπληρωθούν.

Η συμπλήρωση μπορεί να γίνει μετρώντας από την αρχή της γραμμής, συμπληρώνοντας από το τέλος τη διαφορά του κάθε στοιχείου. Αντίστοιχα, μπορεί να γίνει η ίδια καταμέτρηση και από το τέλος, συμπληρώνοντας τα αρχικά σημεία των στοιχείων. Η τεχνική αυτή μπορεί να εφαρμοστεί για όλες τις γραμμές και στήλες για μια αρχική προσέγγιση και εύρεση γρήγορων στοιχείων. (Σχήμα 4.12)

4.3 Προβλήματα NP-πληρότητας (NP-completeness)

4.3.1 Τι είναι τα NP προβλήματα

Τα προβλήματα NP (από τα αρχικά των αγγλικών λέξεων Nondeterministic Polynomial time) είναι μια κατηγορία προβλημάτων στη θεωρία υπολογιστικής πολυπλοκότητας, για τα οποία η ορθότητα μιας δοθείσας λύσης μπορεί να επαληθευτεί σε πολυωνυμικό χρόνο. Αυτό σημαίνει ότι, εάν κάποιος προσφέρει μια πιθανή λύση σε ένα τέτοιο πρόβλημα, μπορούμε να την ελέγξουμε γρήγορα (με έναν «αποτελεσματικό» αλγόριθμο) για να διαπιστώσουμε αν πράγματι ικανοποιεί όλες τις απαιτούμενες συνθήκες.

Για παράδειγμα, σκεφτείτε το κλασικό πρόβλημα 3-SAT, όπου έχουμε μια λογική έκφραση (με μεταβλητές που μπορούν να πάρουν τιμές Αληθές/Ψευδές) και θέλουμε να βρούμε εάν υπάρχει τρόπος να αντιστοιχίσουμε τιμές στις μεταβλητές, ώστε η έκφραση να βγαίνει αληθής. Αν κάποιος μας δώσει απευθείας μία έτοιμη αντιστοίχιση τιμών στις μεταβλητές, μπορούμε σχετικά εύκολα και γρήγορα να ελέγξουμε αν όντως ικανοποιείται η έκφραση. Αυτή η δυνατότητα ελέγχου σε πολυωνυμικό χρόνο αποτελεί βασική ιδιότητα των προβλημάτων NP.

4.3.2 NP-πλήρη Προβλήματα

Μέσα στην κλάση NP, υπάρχει μια υποκατηγορία που ονομάζεται NP-πλήρη (NP-complete) προβλήματα. Αυτά τα προβλήματα είναι τα «δυσκολότερα» στην NP, υπό την έννοια ότι κάθε άλλο πρόβλημα στην NP μπορεί να «αναχθεί» σε κάποιο από αυτά σε πολυωνυμικό χρόνο. Αν βρεθεί (κάποτε) ένας πολυωνυμικός αλγόριθμος για έστω ένα NP-πλήρες πρόβλημα, αυτόματα θα μπορούμε να επιλύουμε σε πολυωνυμικό χρόνο και όλα τα υπόλοιπα προβλήματα της NP. Μέχρι σήμερα, ωστόσο, το ζήτημα αν P=NP παραμένει ανοιχτό και είναι ένα από τα σημαντικότερα άλυτα προβλήματα των μαθηματικών και της πληροφορικής.

4.3.3 NP-δύσκολα (NP-hard) Προβλήματα

Στη θεωρία της υπολογιστικής πολυπλοκότητας, ένα πρόβλημα ονομάζεται NP-δύσκολο, όταν για κάθε πρόβλημα που μπορεί να λυθεί σε πολυωνυμικό χρόνο, η λύση του μπορεί να μην επαληθεύεται σε πολυωνυμικό χρόνο. Τα NP-δύσκολα προβλήματα είναι τόσο δύσκολα και απαιτητικά όσο τα NP-πλήρη προβλήματα, χωρίς όμως να είναι απαραίτητα στην κλάση NP.

4.3.4 Γιατί τα Εικονόσταυρα είναι NP-πλήρη

Τα εικονόσταυρα είναι ενδιαφέροντα προβλήματα λόγω της πολυπλοκότητάς τους και του τρόπου με τον οποίο εμπίπτουν στην κατηγορία των NP-πλήρη προβλημάτων [6] [7]. Για να κατανοήσουμε πλήρως γιατί τα εικονόσταυρα είναι NP-πλήρη, πρέπει να εξετάσουμε δύο σημεία:

- Γιατί ανήκουν στην κατηγορία NP.
- Γιατί είναι NP-δύσκολα.

4.3.4.1 Τα Εικονόσταυρα είναι στην κλάση NP

Ένα πρόβλημα ανήκει στην κατηγορία NP όταν μια προτεινόμενη λύση του μπορεί να επαληθευτεί σε πολυωνυμικό χρόνο. Στην περίπτωση των εικονόσταυρων, αν κάποιος μας δώσει μια πλήρη λύση, μπορούμε να επιβεβαιώσουμε την ορθότητά της σε εύλογο χρόνο. Η διαδικασία επαλήθευσης περιλαμβάνει τον έλεγχο των συμπληρωμένων γραμμών και στηλών σε σχέση με τα στοιχεία (clues) που δίνονται.

4.3.4.2 Τα Εικονόσταυρα είναι NP-hard

Για να αποδειχθεί ότι τα εικονόσταυρα είναι NP-hard, απαιτείται αναγωγή από ένα γνωστό NP-πλήρες πρόβλημα, όπως η 3-SAT. Με αυτήν την αναγωγή, μπορούμε να μετατρέψουμε τις συνθήκες ενός άλλου προβλήματος στις ενδείξεις ενός εικονόσταυρου.

Η αναγωγή αυτή δείχνει ότι οποιοδήποτε NP-πλήρες πρόβλημα μπορεί να μετατραπεί σε ένα εικονόσταυρο μέσα σε πολυωνυμικό χρόνο. Άρα, εάν υπήρχε ένας πολυωνυμικός αλγόριθμος για την επίλυση οποιουδήποτε εικονόσταυρου, τότε θα μπορούσαμε να λύσουμε και όλα τα άλλα NP-πλήρη προβλήματα χρησιμοποιώντας την ίδια διαδικασία.

Αυτό σημαίνει ότι τα εικονόσταυρα δεν είναι μόνο στην NP, αλλά είναι και NP-hard, γεγονός που τα καθιστά NP-πλήρη.

4.3.4.3 Συμπεράσματα

Τα εικονόσταυρα είναι NP-πλήρη επειδή:

- Η ορθότητα μιας προτεινόμενης λύσης μπορεί να επαληθευτεί σε πολυωνυμικό χρόνο.
- Κάθε πρόβλημα στην κατηγορία NP μπορεί να μειωθεί σε ένα εικονόσταυρο σε πολυωνυμικό χρόνο.

Η πολυπλοκότητα των εικονόσταυρων εξηγεί γιατί απαιτούνται προσεγγιστικές ή ευρετικές μέθοδοι για την επίλυσή τους, ιδίως όταν πρόκειται για μεγάλα ή πολύπλοκα παζλ.

4.4 Σύνοψη

Η παρούσα ενότητα παρουσίασε τις διάφορες μεθοδολογίες επίλυσης εικονόσταυρων, ξεκινώντας από βασικές τεχνικές όπως η επίλυση ανά γραμμή, έως πιο σύνθετες τεχνικές που περιλαμβάνουν την επίλυση με πολλαπλές γραμμές, εκτιμήσεις και μαθηματικές πράξεις.

Επιπλέον, έγινε συζήτηση για την ένταξη των εικονόσταυρων στα NP-πλήρη προβλήματα, υπογραμμίζοντας την υπολογιστική πολυπλοκότητά τους και την ανάγκη για αποτελεσματικούς αλγόριθμους επίλυσης. Τέλος, οι προκλήσεις που συνδέονται με την επίλυση τέτοιων παζλ καταδεικνύουν την ανάγκη για περαιτέρω έρευνα και ανάπτυξη πιο προηγμένων μεθόδων και εργαλείων.

Κεφάλαιο 5ο: Οι επιστημονικές και τεχνολογικές εξελίξεις στην επίλυση γρίφων

5.1 Εισαγωγή

Το παρόν κεφάλαιο εμβαθύνει στις ευρύτερες επιστημονικές και τεχνολογικές εξελίξεις που αφορούν την επίλυση γρίφων. Τέτοιου είδους εξελίξεις είναι η τεχνητή νοημοσύνη, η μηχανική μάθηση, οι αλγόριθμοι βελτιστοποίησης και η θεωρία υπολογιστικής πολυπλοκότητας. Επιπλέον, εξελίσσονται και αναπτύσσονται εφαρμογές που διευκολύνουν στην επίλυση γρίφων με ένα διαδραστικό και φιλικό τρόπο προς το χρήστη.

Ένα προϊόν (π.χ. εφαρμογή) δημιουργείται διότι υπάρχει ανάγκη επίλυσης ενός προβλήματος. Απαραίτητη προϋπόθεση για τη δημιουργία του είναι η ύπαρξη του προβλήματος καθαυτού. Αφού εντοπιστεί το πρόβλημα που χρήζει επίλυσης, γίνεται εκτενής έρευνα και ανάλυση των ήδη υπαρχόντων προϊόντων (π.χ. εφαρμογών) που προσέφεραν λύση.

Το βασικότερο και κυριότερο πρόβλημα που πραγματεύεται και δύναται να δώσει λύση αυτή η εργασία αφορά στις ελλείψεις που υπάρχουν στους ήδη υπάρχοντες λύτες. Παρακάτω αναφέρονται και περιγράφονται όλα αυτά τα προγράμματα - λύτες εικονόσταυρων καθώς και θα τονισθεί ποιες ανάγκες καλύπτει ή όχι το καθένα από αυτά.

5.2 Ο ”pbn solve” Λύτης Παζλ Ζωγραφικής με Αριθμούς του Γιάν Γουόλτερ (Jan Wolter)

Ο Γιάν Γουόλτερ (Jan Wolter) κατέχει έναν από τους πιο γνωστούς λύτες εικονόσταυρων, το PBN (Picture-By-Number) Solve [8]. Βασισμένος στην εμπεριστατωμένη του έρευνα σε σχέση με τα εικονόσταυρα [9], κατάφερε να χτίσει έναν δικό του αλγόριθμο λύσης εικονοσταύρων. Έχει δημιουργήσει μια ιστοσελίδα που, εκτός του λύτη, περιλαμβάνει μια γκάμα παζλ και ένα σύστημα παικτών μέσα στο οποίο οι χρήστες μπορούν να φτιάχνουν, να λύνουν και να σχολιάζουν παζλ της ιστοσελίδας, λέγοντας και τις απόψεις τους και δημιουργώντας μια συζήτηση μεταξύ τους. Αν και στην αρχική σελίδα της ιστοσελίδας του αναφέρεται πως ο Γιάν Γουόλτερ απεβίωσε το 2015, η ιστοσελίδα του παραμένει ολοζώντανη ακόμα και σήμερα με πλήθος χρηστών να την επισκέπτονται καθημερινά.

5.2.1 Χαρακτηριστικά

Μερικά από τα σημαντικότερα χαρακτηριστικά του λύτη του Γιάν Γουόλτερ είναι τα παρακάτω:

1. Είναι πρόγραμμα ανοιχτού κώδικα με άδεια Apache 2.0. Αυτό σημαίνει πως ο οποιοσδήποτε μπορεί να διαβάσει και να επεξεργαστεί τον κώδικα του.
2. Είναι γραμμένο στη γλώσσα C για συστήματα Unix, μια πολύ καλή γλώσσα για πραγματοποίηση πολλαπλών μαθηματικών πράξεων διότι προσφέρει γρήγορη επεξεργασία τιμών.
3. Χρησιμοποιεί ευέλικτη μορφή XML για την είσοδο των παζλ, αλλά μπορεί να διαβάσει διάφορες άλλες μορφές αρχείων, καθώς και μορφές αρχείων από άλλους λύτες.
4. Λύνει πολύχρωμα παζλ καθώς και παραδοσιακά ασπρόμαυρα παζλ.

5. Δεν υπάρχει έμφυτο όριο στο μέγεθος των παζλ ή στον αριθμό των χρωμάτων.

6. Χρησιμοποιεί συγκεκριμένο αριθμό μνήμης, ο οποίος παραμένει σταθερός ανεξάρτητα από τη διάρκεια και τις απαιτήσεις λειτουργίας του προγράμματος.

7. Χρησιμοποιεί μόνο λογική γραμμών και χρωμάτων για την επίλυση, και καταφέρνει να φέρει γρήγορα και αξιόπιστα αποτελέσματα.

8. Μπορεί να εκτελεστεί από τη γραμμή εντολών, αλλά μπορεί επίσης να λειτουργήσει ως πρόγραμμα CGI, διαβάζοντας τα επιχειρήματά του από CGI και εξάγοντας τα αποτελέσματα σε μορφή XML, ώστε να μπορεί να χρησιμοποιηθεί ως εφαρμογή AJAX.

9. Η αρχιτεκτονική του λύτη σχεδιάστηκε για να μπορεί να διαχειρίζεται γρίφους ”triddler” (εξαγωνικοί γρίφοι με τριγωνικά κελιά και τρία σύνολα στοιχείων), αν και υπάρχουν πολλά θεμέλια για να υποστηριχθεί αυτό, απέχει πολύ από την πλήρη υλοποίηση.

5.2.2 Ελλείψεις

Με την πάροδο του χρόνου και την εξέλιξη των απαιτήσεων μπορεί να αναγνωριστεί πως το πρόγραμμα θα μπορούσε να αξιοποιήσει κάποιες βελτιώσεις ώστε να είναι ακόμα και τώρα επίκαιρο.

1. Προσβάσιμο και ανασχεδιασμένο UI, έτσι ώστε να μη χρειάζεται το πρόγραμμα λογισμικό ώστε να “τρέξει” και να είναι πιο εύχρηστο και φιλικό.

2. Προσβάσιμη και περισσότερο επεξηγηματική ανάλυση ως προς τη λειτουργία του προγράμματος, όχι μόνο σε θεωρητικό επίπεδο.

3. Προβολή βημάτων λύσης στον χρόνο που επιθυμεί ο χρήστης, καθώς και προσαρμογή δυσκολίας με βάση τις ανάγκες του κάθε χρήστη.

4. Ανάπτυξη πιο αποδοτικών αλγορίθμων για την επίλυση πολυπλοκότερων παζλ.

5. Παροχή οδηγιών για νέους χρήστες.

6. Χρήση τεχνητής νοημοσύνης για πρόβλεψη και επίλυση παζλ.

5.2.3 Ανάλυση κώδικα λύτη

Ο λύτης Pbnsolve χρησιμοποιεί έναν συνδυασμό εξελιγμένων τεχνικών για να αντιμετωπίσει την πρόκληση της επίλυσης παζλ εικονόσταυρων. Κάθε μέθοδος συμβάλλει μοναδικά στη συνολική του ικανότητα, εξασφαλίζοντας ανθεκτικότητα και αποδοτικότητα σε διάφορα επίπεδα δυσκολίας.

Παρακάτω ακολουθούν οι βασικές τεχνικές και οι επιπτώσεις τους.

5.2.3.1 Αλγόριθμοι εύρεσης κελιών

Οι αλγόριθμοι αυτοί επικεντρώνονται στην αναζήτηση λύσεων για τα κελιά του παζλ. Αναλύουν γραμμές και κελιά με στόχο την συμπλήρωσή τους.

5.2.3.1.1 Λύση ανά γραμμή

Αυτή η μέθοδος αποτελεί μια συνδυασμένη προσέγγιση των λογικών αλγορίθμων που αναλύθηκαν συγκεκριμένα της «Επικάλυψη», «Αδυνατότητας πρόσβασης» και «Δεν χωράει». Οι παραπάνω αλγόριθμοι χρησιμοποιούν την ίδια αρχή, δηλαδή αξιοποιούν την πληροφορία μιας γραμμής ή στήλης κάθε φορά, ενώ είτε δεν έχουν καθόλου συμπληρωμένα κελιά είτε τα συμπληρωμένα κελιά είναι γνωστά ακριβώς σε ποια στοιχεία αντιστοιχούν.

Αυτή η προσέγγιση ξεκινάει εντοπίζοντας τη λύση που τοποθετεί όλα τα στοιχεία όσο το δυνατόν πιο αριστερά και τη λύση που τα τοποθετεί όσο το δυνατόν πιο δεξιά για κάθε γραμμή, και στη συνέχεια τέμνει αυτές τις λύσεις για να εντοπίσει αν κάποιο ή κάποια κελιά είναι είτε χρωματισμένα είτε κενά.

Ενώνει τις δύο λύσεις χρωματίζοντας κάθε κελί ίδιου στοιχείου που είναι και στις δύο λύσεις, και διαγράφει ως κενό οποιοδήποτε κελί βρίσκεται μεταξύ του ίδιου ζεύγους στοιχείου.

Παρότι αυτή η προσέγγιση είναι γρήγορη, είναι ατελής και συχνά απαιτεί επιπλέον βήματα για να σημειώσει όλα τα πιθανά κελιά. Είναι ιδιαίτερα αποτελεσματική στα αρχικά στάδια του παζλ για να λύσει γρήγορα απλούστερα τμήματα.

5.2.3.1.2 Μάντεμα

Αυτός ο αλγόριθμος βασίζεται στη μεθοδολογία της «Μέθοδος Υπόθεσης». Δηλαδή, αρχίζει με μια υπόθεση τοποθετώντας μια λύση στο πλέγμα, με σκοπό είτε να λυθεί το παζλ, είτε να οδηγηθεί σε ένα άτοπο αποτέλεσμα. Έτσι, επιβεβαιώνει ότι το στοιχείο που δοκίμασε έχει την αντίθετη τιμή από την τιμή της δοκιμής, ή αν το παζλ δεν λυθεί, να χρειαστεί να κάνει μια άλλη μαντεψιά.

Αυτό γίνεται κρατώντας μια λίστα ”ιστορικού αναίρεσης”, όπου μετά την τοποθέτηση της υπόθεσης ενός κελιού, καταγράφονται εκεί όλες οι συμπληρώσεις κελιών από την επίλυση γραμμής. Αν το παζλ λυθεί, τότε μπορούμε να σταματήσουμε - ο μόνος λόγος για να συνεχίσουμε είναι αν θέλουμε να επιβεβαιώσουμε ότι το παζλ έχει περισσότερες από μία λύσεις. Όταν βρεθεί μια αντίφαση, αναιρούνται όλες οι αλλαγές και γνωρίζουμε πλέον ότι το κελί που συμπληρώθηκε ως υπόθεση δεν περιέχει την τιμή που του δόθηκε. Αν το παζλ ούτε λυθεί ούτε οδηγηθεί σε αντίφαση, μπορούμε να κάνουμε μια άλλη μαντεψιά, κάτι που αν γίνει επανειλημμένα μπορεί να αυξήσει την πολυπλοκότητα του παζλ σημαντικά και να καταλήξει αναποτελεσματικό.

Ο Γιάν αναφέρει ότι κάποιες μαντεψιές προωθούν τη λύση πιο γρήγορα από άλλες, οπότε δεν είναι απαραίτητο ο αλγόριθμος να αναλώνεται σε πολλαπλές απανωτές μαντεψιές. Αντί για μαντεψιές πάνω στις μαντεψιές, συνιστάται να γίνονται εξ ολοκλήρου νέες μαντεψιές.

5.2.3.2 Αλγόριθμοι επιλογής γραμμών προς επίλυση

Οι παρακάτω μέθοδοι δεν επικεντρώνονται απευθείας στη λύση του παζλ, αλλά στους τρόπους συλλογής γραμμών με σκοπό να χρησιμοποιηθούν ως είσοδος στους παραπάνω αλγόριθμους εύρεσης κελιών.

5.2.3.2.1 Λογική Επίλυση

Αυτή η συγκεκριμένη μέθοδος δεν επικεντρώνεται απευθείας στη λύση του παζλ, αλλά στη συλλογή των γραμμών προς ανάλυση και την παράδοσή τους στον αλγόριθμο «Λύση ανά γραμμή», όπως περιγράφηκε προηγουμένως.

Αυτός ο αλγόριθμος εφαρμόζει ένα σύστημα επίλυσης, το οποίο βασίζεται σε μια λίστα εργασιών που λειτουργεί σαν ένα είδος οδηγού. Μέσω αυτής της λίστας, επιλέγονται συστηματικά γραμμές και στήλες για να εξεταστούν. Κατά την επίλυση κάθε γραμμής, ο αλγόριθμος ενημερώνει τις διασταυρούμενες γραμμές με βάση τα νέα σημειωμένα κελιά, προκειμένου να προχωρήσει σε επιπλέον λύσεις. Σε πολλές περιπτώσεις, ο αλγόριθμος μπορεί να καταφέρει να επιλύσει ολόκληρο το παζλ, εάν οι λογικοί περιορισμοί είναι επαρκείς.

Αν το παζλ δεν λυθεί, τότε ο αλγόριθμος ακολουθεί έναν εξαντλητικό έλεγχο για να διαπιστώσει εάν κάποιο από τα κελιά που έμειναν μπορούν να συμπληρωθούν. Αυτός ο έλεγχος γίνεται παίρνοντας ξανά μία-μία τις γραμμές, και γίνεται τοποθέτηση όλων των πιθανών χρωμάτων που μπορεί να πάρει το κελί (μόνο ένα χρώμα αν έχουμε ασπρόμαυρο παζλ). Στην περίπτωση που η λογική επίλυση αποτύχει να επιλύσει το παζλ, ο αλγόριθμος προσπαθεί εναλλακτικές στρατηγικές, όπως το μάντεμα ή η ανίχνευση, προκειμένου να βρει μια εναλλακτική λύση.

5.2.3.2.2 Ανίχνευση

Η «Ανίχνευση» είναι συνοπτικά μια πιο στοχευμένη εφαρμογή του «Μαντέματος». Το pbnsolve χρησιμοποιεί την προσέγγιση ανίχνευσης ως την προεπιλεγμένη μέθοδό του. Σε αντίθεση με τις παραδοσιακές μεθόδους που κάνουν μαντεψιές βάσει απλών κανόνων, η προσέγγιση ανίχνευσης δημιουργεί μια εκτενή λίστα πιθανών μαντεψιών και αξιολογεί καθεμία από αυτές συστηματικά. Αυτή η διαδικασία περιλαμβάνει την εξέταση των πιθανών λύσεων χωρίς να δεσμεύεται άμεσα σε αυτές, επιτρέποντας έτσι μια πιο στοχευμένη επιλογή του επόμενου βήματος.

Η προσέγγιση ανίχνευσης λειτουργεί επιλέγοντας ένα σύνολο κελιών που είναι πιθανοί υποψήφιοι για μαντεψιά. Αυτό το σύνολο προσδιορίζεται βάσει δύο κριτηρίων:

Κελιά Γειτονικά με Πρόσφατα Αλλαγμένα Κελιά: Ο αλγόριθμος επιλέγει κάθε άλυτο κελί που είναι γειτονικό με ένα κελί του οποίου η κατάσταση έχει αλλάξει από την τελευταία μαντεψιά. Αυτό το κριτήριο διασφαλίζει ότι η εστίαση παραμένει σε περιοχές του παζλ όπου οι πρόσφατες αλλαγές μπορεί να παρέχουν νέες πληροφορίες.

Κελιά με Πολλαπλούς Λυμένους Γείτονες: Επιπλέον, ο αλγόριθμος περιλαμβάνει κάθε άλυτο κελί που έχει δύο ή περισσότερα γειτονικά κελιά που έχουν λυθεί. Τα κελιά στις άκρες του πλέγματος θεωρούνται επίσης λυμένα σε αυτό το πλαίσιο. Αυτό το κριτήριο βοηθά στον εντοπισμό κελιών που είναι πιο πιθανό να επηρεαστούν από τα γύρω λυμένα κελιά, αυξάνοντας έτσι τις πιθανότητες μιας σωστής μαντεψιάς.

Αφού προσδιοριστεί το σύνολο των υποψήφιων κελιών, ο αλγόριθμος εξετάζει κάθε πιθανή μαντεψιά για αυτά τα κελιά. Εκτελεί τον λογικό αλγόριθμο για κάθε μαντεψιά για να αξιολογήσει τον αντίκτυπό της στο παζλ. Η μαντεψιά που οδηγεί στην περισσότερη πρόοδο προς την επίλυση του παζλ επιλέγεται ως το επόμενο βήμα.

Συμπεράσματα

Η προσέγγιση ανίχνευσης προσφέρει διάφορα πλεονεκτήματα:

- **Ενημερωμένη Λήψη Αποφάσεων:** Με την αξιολόγηση κάθε πιθανής μαντεψιάς πριν δεσμευτεί, ο αλγόριθμος μειώνει την πιθανότητα να κάνει λανθασμένες μαντεψιές που θα μπορούσαν να οδηγήσουν σε αδιέξοδα.
- **Αποτελεσματικότητα σε Δύσκολα Παζλ:** Αυτή η μέθοδος είναι ιδιαίτερα αποτελεσματική σε πιο περίπλοκα παζλ όπου οι απλές λογικές τεχνικές μπορεί να αποτύχουν. Χρησιμοποιεί τις δυνατότητες του λογικού αλγορίθμου για να κάνει πιο σωστές μαντεψιές.
- **Προσαρμοστικότητα:** Η προσέγγιση ανίχνευσης είναι προσαρμοστική, συνεχώς βελτιώνοντας το σύνολο των υποψήφιων κελιών βάσει της εξελισσόμενης κατάστασης του παζλ, διατηρώντας έτσι την εστίαση στις πιο υποσχόμενες περιοχές.

Συνοψίζοντας, η προσέγγιση ανίχνευσης στο pbnsolve αντιπροσωπεύει μια εξελιγμένη και προσαρμοστική μέθοδο για την επίλυση παζλ εικονόσταυρων. Με τη δημιουργία και αξιολόγηση μιας εκτενούς λίστας πιθανών μαντεψιών, ο αλγόριθμος βελτιώνει την ικανότητά του να παίρνει ενημερωμένες αποφάσεις, ενισχύοντας έτσι τη συνολική αποτελεσματικότητα και ακρίβεια της διαδικασίας επίλυσης.

5.2.3.3 Μέθοδοι συνδυασμού αλγορίθμων

Τελευταία στην κατηγορία των μεθοδολογιών επίλυσης βρίσκονται οι αλγόριθμοι που επιδιώκουν την εύρεση κατάλληλων συνδυασμών των προηγούμενων κατηγοριών λύσεων.

5.2.3.3.1 Βάδισμα και Τρέξιμο

Η πρώτη συνένωση αφορά τη σύμπλεξη της μεθόδου ”Μάντεμα” με αυτή της ”Ανίχνευσης”. Μέσω πειραμάτων διαπιστώθηκε ότι, παρά την θεωρητική αποδοτικότητα της μεθόδου ”Ανίχνευσης”, στην πράξη η μέθοδος ”Μάντεμα” επιλύει το πρόβλημα με μεγαλύτερη αποδοτικότητα, ακόμη και σε ποσοστό που ξεπερνάει το 50%.

Αυτό συμβαίνει κυρίως σε παζλ με πολλαπλές λύσεις ή υψηλή πολυπλοκότητα, όπου η καταγραφή όλων των πιθανών μονοπατιών λύσης απαιτεί πολύ χρόνο. Έτσι, είναι πιο αποδοτικό να ακολουθηθεί ένα ή δύο μονοπάτια με σκοπό την εύρεση μιας λύσης ή τον εντοπισμό μιας αντίφασης.

5.2.3.3.2 Συγχώνευση

Η συγχώνευση αποσκοπεί στο να συνδυάσει τις λύσεις από όλους τους αλγόριθμους επίλυσης, για κάθε πιθανή τιμή που μπορεί να λάβει ένα κελί. Η ιδέα είναι ότι αν ένα κελί έχει την ίδια τιμή σε όλες τις πιθανές λύσεις που προκύπτουν από τη γραμμή του, τότε μπορούμε να το συμπληρώσουμε με αυτήν την τιμή.

Ο Γιάν σημειώνει ότι, αν και ο αλγόριθμος λειτουργεί σωστά, δεν βελτιώνει την απόδοση του προγράμματος· αντίθετα, καταναλώνει περισσότερη μνήμη, καθώς απαιτεί την αποθήκευση όλων των τιμών για

σύγκριση. Για αυτόν τον λόγο, η συγχώνευση συνήθως δεν ενεργοποιείται αυτόματα στον λύτη, αλλά μόνο εάν επιλεγεί χειροκίνητα.

5.2.3.3.3 Διαχείριση μνήμης

Ένας λύτης χρειάζεται και χρησιμοποιεί πολύ μνήμη σε ένα σύστημα, για αυτόν τον λόγο είναι μόνο λογικό να πέσει θέμα συζήτησης η καλύτερη διαχείριση αυτής. Ο λύτης χρειάζεται να αποθηκεύει και να κρατάει πολλαπλά δεδομένα στην μνήμη του, όπως την ενεργή λύση του παζλ, τις πολλαπλές λίστες με τις προσωρινές λύσεις, και το τρέξιμο του ίδιου του αλγορίθμου για την εύρεση αυτών των λύσεων.

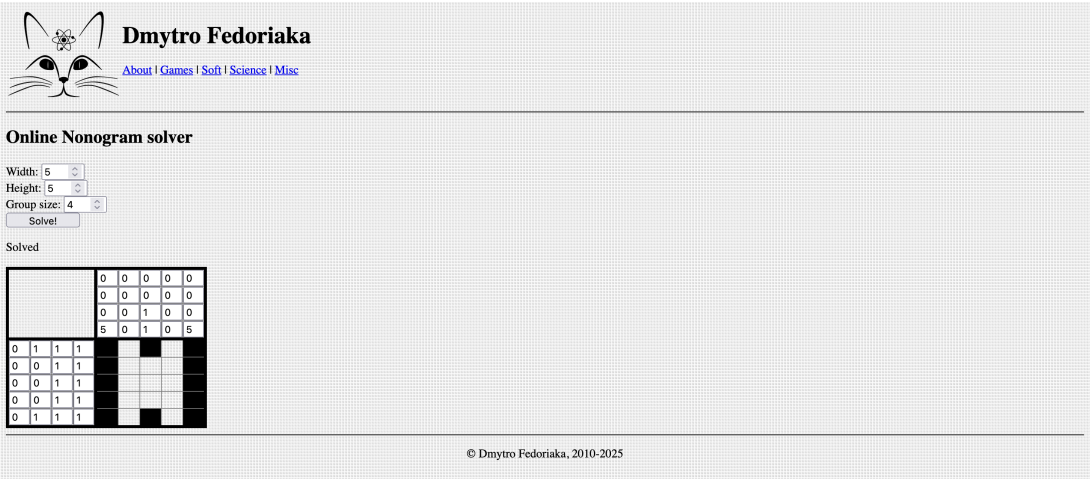
5.3 Ο διαδικτυακός λύτης του Δμίτρο Φεδωριάκα (Dmytro Fedoriaka)

5.3.1 Εισαγωγή

Ο διαδικτυακός λύτης του Δμίτρο Φεδωριάκα [4] διατίθεται μέσω μίας ευχάριστης ιστοσελίδας, η οποία χαρακτηρίζεται από εντυπωσιακά χρώματα και απλό σχεδιασμό, διευκολύνοντας τον επισκέπτη να κατανοήσει το περιβάλλον χειρισμού. Σε εμφανές σημείο της σελίδας επιτρέπεται η ρύθμιση βασικών παραμέτρων, όπως το μέγεθος του εικονόσταυρου και οι αριθμοί (στοιχεία) που αφορούν κάθε γραμμή ή στήλη.

Η εισαγωγή των στοιχείων γίνεται σε επιμέρους πεδία, επιτρέποντας την προσαρμογή του παζλ σε διάφορες διαστάσεις. Αυτή η δυνατότητα βοηθά τον χρήστη να ορίσει με ακρίβεια το σχήμα και τις απαιτήσεις του εικονόσταυρου.

Παρόλο που ο σχεδιασμός της σελίδας είναι συνολικά προσεγμένος, η διαμόρφωση του παζλ σε μεμονωμένα κελιά παραμένει αρκετά λιτή. Αυτό σημαίνει ότι ο χρήστης βλέπει στην οθόνη τις πληροφορίες για τα «διαστήματα» του εικονόσταυρου (τις ομάδες των γεμάτων κελιών που πρέπει να υπάρχουν σε κάθε γραμμή και στήλη), χωρίς ωστόσο να παρέχονται πολλές οδηγίες για την ακριβή διαδικασία εισαγωγής τους. Η σελίδα φαίνεται να απευθύνεται κυρίως σε χρήστες που ήδη γνωρίζουν τα βασικά των εικονόσταυρων.



Σχήμα 5.1: Λυμένο εικονόσταυρο από την σελίδα του Δμίτρο Φεδωριάκα (Dmytro Fedoriaka). [4]

5.3.2 Θετικά εφαρμογής

Ορισμένα από τα βασικά πλεονεκτήματα του διαδικτυακού λύτη του Δμίτρο Φεδωριάκα είναι:

- **Ευκολία χρήσης:** Ο λύτης λειτουργεί εξ ολοκλήρου διαδικτυακά, απαλλάσσοντας τον χρήστη από την ανάγκη εγκατάστασης προγράμματος στον υπολογιστή του.
- **Ευελιξία πρόσβασης:** Η υπηρεσία μπορεί να χρησιμοποιηθεί από οποιονδήποτε σύγχρονο φυλλομετρητή (browser), διευκολύνοντας τη χρήση της σε διάφορες συσκευές και λειτουργικά συστήματα.
- **Κατανοητά μηνύματα:** Παρέχει ανατροφοδότηση σχετικά με την κατάσταση του παζλ, όπως εάν έχει πολλαπλές λύσεις ή εάν οι ενδείξεις είναι ασυμβίβαστες.

5.3.3 Αρνητικά εφαρμογής

Παρά τα πλεονεκτήματά του, ο διαδικτυακός λύτης παρουσιάζει και ορισμένες αδυναμίες:

- **Περιορισμένη φιλικότητα χρήσης:** Η χειροκίνητη συμπλήρωση των κελιών δεν είναι ιδιαίτερα πρακτική, ειδικά για μεγάλα πλέγματα.
- **Έλλειψη διαδραστικότητας:** Δεν προσφέρει δυνατότητα προβολής της λύσης βήμα προς βήμα, κάτι που θα ήταν χρήσιμο για εκπαιδευτικούς σκοπούς.

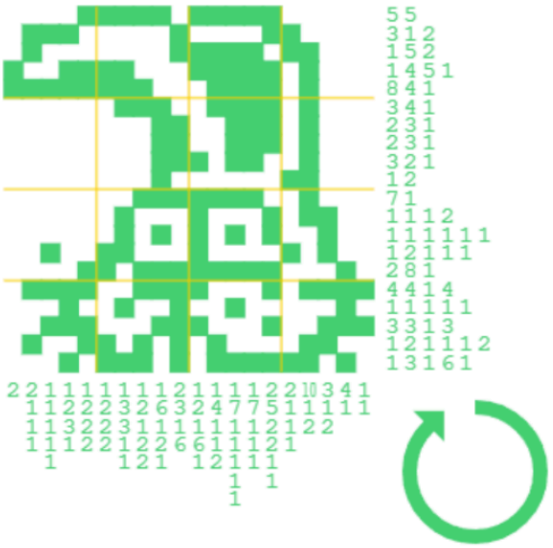
5.3.4 Συμπεράσματα

Ο διαδικτυακός λύτης του Δμίτρο Φεδωριάκα προσφέρει μια γρήγορη και εύχρηστη λύση για χρήστες που θέλουν να επιβεβαιώσουν την ορθότητα ενός εικονόσταυρου χωρίς την ανάγκη εγκατάστασης λογισμικού. Ωστόσο, οι περιορισμοί του, όπως η έλλειψη φιλικότητας στη συμπλήρωση κελιών και η απουσία προβολής βημάτων λύσης, μειώνουν τη χρηστικότητα του για πιο απαιτητικούς χρήστες.

5.4 Ο διαδικτυακός λύτης του Ζου Τσι (Zhou Qi)

5.4.1 Εισαγωγή

Η ιστοσελίδα του Ζου Τσι (Zhou Qi) αποτελεί μια καλοσχεδιασμένη και λιτή εφαρμογή, με βασικό προσανατολισμό τους *προγραμματιστές*. Περιλαμβάνει γραμμές κώδικα και παραδείγματα, δίνοντας τη δυνατότητα εκτέλεσης του λύτη *live* μέσα από τη σελίδα. Ο χρήστης μπορεί να παρακολουθήσει βήμα προς βήμα τη διαδικασία επίλυσης, ενώ παρέχονται τρεις βασικές λειτουργίες: **Nonogram Editor** για τη δημιουργία νέων εικονόσταυρων, **Nonogram Game** όπου ο χρήστης μπορεί να λύσει παζλ (χωρίς την παροχή άμεσων διορθώσεων λαθών, αλλά με ένδειξη επιτυχίας στο τέλος), και **Nonogram Solver**, όπου ο λύτης επιλύει το παζλ που έχει καταχωρηθεί, εμφανίζοντας την πρόοδο σε πραγματικό χρόνο.



Σχήμα 5.2: Λυμένο εικονόσταυρο από την σελίδα του Ζου Τσι (Zhou Qi). [5]

5.4.2 Λειτουργία λύτη

Από τη χρήση και παρατήρηση της εφαρμογής, διαπιστώνεται ότι ο αλγόριθμος επιλύει το παζλ περνώντας διαδοχικά όλες τις γραμμές και στήλες, επαναλαμβάνοντας τη διαδικασία μέχρι να ολοκληρωθεί η επίλυση. Καθώς ο κώδικας που διατίθεται στο *GitHub* δεν περιλαμβάνει σχόλια ή *documentation*, η κατανόηση της ακριβούς μεθοδολογίας είναι περιορισμένη. Επιπλέον, η εφαρμογή δεν επιτρέπει την παύση ή την ανάλυση συγκεκριμένων βημάτων, κάτι που καθιστά δύσκολη τη λεπτομερή μελέτη της διαδικασίας μέσω *reverse engineering*.

5.4.3 Θετικά

- **Απλότητα**: Καλαίσθητη και λιτή εφαρμογή, με *one-page* σχεδίαση, που επικεντρώνεται αποκλειστικά στις βασικές λειτουργίες.
- **Οπτική επίδειξη**: Εμφανίζει γραφικά τη διαδικασία επίλυσης, διευκολύνοντας την παρακολούθηση της προόδου του λύτη.

5.4.4 Αρνητικά

- **Έλλειψη τεκμηρίωσης**: Ο κώδικας και η λειτουργία του λύτη δεν συνοδεύονται από επαρκές *documentation*, περιορίζοντας την κατανόηση της μεθοδολογίας.
- **Μη επαρκής μελέτη βημάτων**: Η εφαρμογή δεν επιτρέπει τη λεπτομερή παρακολούθηση των ενδιάμεσων βημάτων της επίλυσης.
- **Δυσχέρειες δημιουργίας παζλ**: Η εισαγωγή δεδομένων για το παζλ γίνεται μέσω *text fields*, απαιτώντας χειροκίνητη καταχώρηση ανά γραμμή ή στήλη. Δεν παρέχεται λειτουργία «ζωγραφικής» για την ευκολότερη δημιουργία εικονόσταυρων.

5.4.5 Σύνοψη

Η ιστοσελίδα του Zhou Qi αποτελεί ένα καλαίσθητο και εύχρηστο εργαλείο που προσφέρει τρεις βασικές λειτουργίες για τους εικονόσταυρους. Αν και ο λύτης διαθέτει δυνατότητες γραφικής επίδειξης της προόδου, η έλλειψη τεκμηρίωσης και η περιορισμένη αλληλεπίδραση στα βήματα επίλυσης μειώνουν την ευκολία βαθύτερης μελέτης και χρήσης του. Παρά τις ελλείψεις, παραμένει ένα εργαλείο με προοπτική βελτίωσης, ιδιαίτερα αν εμπλουτιστεί με περισσότερες λειτουργίες και διευκολύνσεις για τη δημιουργία και την κατανόηση της διαδικασίας επίλυσης.

5.5 Προσδιορισμός ελλείψεων στις υπάρχουσες λύσεις

Αυτή η υποενότητα επικεντρώνεται στον εντοπισμό και την ανάλυση των κενών και των περιορισμών που υπάρχουν στις τρέχουσες λύσεις επίλυσης παζλ μη-γραμμάτων. Ειδικότερα, εξετάζεται η έλλειψη προσβασιμότητας, η επεξήγηση των βημάτων λύσης και η προβολή της βήμα προς βήμα, καθώς και η απουσία λύτη στο πλαίσιο του Flutter. Η διερεύνηση αυτών των ζητημάτων βοηθά στην καλύτερη κατανόηση των αναγκών και των προκλήσεων που δύνανται να αντιμετωπίσουν οι χρήστες, προτείνοντας επίσης προσεγγίσεις που θα βελτιώσουν την εμπειρία χρήσης σε όλα τα αναφερόμενα πεδία.

5.5.1 Ελλείψεις

Οι κύριες ελλείψεις των υπαρχόντων αλγορίθμων επίλυσης παζλ μη-γραμμάτων σχετίζονται με την προσβασιμότητα, την επεξήγηση των βημάτων, τη δυνατότητα προβολής των βημάτων χειροκίνητα και την καινοτομία στη χρήση πιο υψηλού επιπέδου γλωσσών όπως το Flutter. Αυτές οι ελλείψεις αναδεικνύουν την ανάγκη για ανάπτυξη πιο φιλικών προς τον χρήστη λύσεων, οι οποίες παρέχουν αναλυτικές επεξηγήσεις και δυνατότητες χειροκίνητης προβολής των βημάτων της λύσης. Με την ενσωμάτωση αυτών των χαρακτηριστικών, οι λύσεις θα γίνουν πιο προσιτές και εκπαιδευτικές, προσφέροντας μια καλύτερη εμπειρία στους χρήστες.

5.5.2 Έλλειψη προσβασιμότητας

Οι περισσότεροι υπάρχοντες αλγόριθμοι επίλυσης παζλ, ιδιαίτερα οι πιο αποδοτικοί και περίπλοκοι, απαιτούν εξειδικευμένες γνώσεις για να χρησιμοποιηθούν. Συχνά, οι λύσεις αυτές διατίθενται μέσω προγραμμάτων που εκτελούνται στη γραμμή εντολών, με τα παζλ να εισάγονται μέσω συγκεκριμένων αρχείων σε καθορισμένη μορφή.

Για έναν προγραμματιστή, η διαδικασία αυτή μπορεί να είναι σχετικά εύκολη, καθώς συνήθως παρέχονται αναλυτικές οδηγίες. Παρόλα αυτά, ακόμη και για έναν έμπειρο προγραμματιστή, η διαδικασία αυτή μπορεί να αποδειχθεί πολύπλοκη και χρονοβόρα, απαιτώντας συγκέντρωση και κατανόηση των οδηγιών.

Η έλλειψη μιας εύχρηστης διεπαφής καθιστά τις εφαρμογές αυτές δύσκολες στη χρήση για τους μη-εξειδικευμένους χρήστες, περιορίζοντας την προσβασιμότητα και την ευκολία χρήσης. Μια πιο φιλική προς τον χρήστη διεπαφή θα μπορούσε να βελτιώσει σημαντικά την εμπειρία χρήσης, κάνοντας τις λύσεις αυτές πιο προσιτές και ευχάριστες για το ευρύ κοινό.

5.5.3 Έλλειψη επεξήγησης

Ένα σημαντικό ποσοστό των υφιστάμενων λύσεων παρουσιάζει άμεσα τα τελικά αποτελέσματα, χωρίς να παρέχει ενδιάμεσες εξηγήσεις. Οι λίγες εφαρμογές που δείχνουν τα βήματα της λύσης, συνήθως το κάνουν με γρήγορα κινούμενα σχέδια, καθιστώντας δύσκολη την κατανόηση της διαδικασίας.

Η απουσία αναλυτικών επεξηγήσεων για τη λογική πίσω από κάθε συμπλήρωση κάνει δύσκολη τη βαθύτερη κατανόηση της διαδικασίας επίλυσης. Αυτό περιορίζει τη δυνατότητα εκμάθησης νέων τεχνικών και στρατηγικών, καθώς οι χρήστες δεν έχουν τη δυνατότητα να δουν και να κατανοήσουν τα ενδιάμεσα βήματα της λύσης.

5.5.4 Έλλειψη προβολής της λύσης χειροκίνητα βήμα προς βήμα

Όπως αναφέρθηκε παραπάνω, λίγες είναι οι εφαρμογές που παρουσιάζουν τα βήματα της λύσης, και καμία δεν προσφέρει τη δυνατότητα προβολής αυτών των βημάτων με χειροκίνητο τρόπο. Οι χρήστες δεν μπορούν να σταματήσουν τη λύση ή να μειώσουν την ταχύτητα εναλλαγής των βημάτων, καθιστώντας δύσκολη την παρακολούθηση και κατανόηση της διαδικασίας.

Επιπλέον, δεν υπάρχει δυνατότητα πλοήγησης μπρος και πίσω στη λύση για τη μελέτη συγκεκριμένων σημείων. Αυτή η έλλειψη περιορίζει την πρόσβαση σε μια ανάλυση της προόδου της λύσης, καθιστώντας δύσκολη την εκμάθηση και την κατανόηση των επιμέρους βημάτων. Μια τέτοια δυνατότητα θα ήταν ιδιαίτερα χρήσιμη για τη σχολαστική μελέτη με σκοπό την κατανόηση των παζλ.

5.5.5 Έλλειψη ενός λύτη στο Flutter framework

Κατά την έρευνα και τη μελέτη που πραγματοποιήθηκε στα πλαίσια της συγγραφής της παρούσας εργασίας, δεν βρέθηκε κανένας λύτης γραμμένος στο πλαίσιο Flutter. Αυτό είναι αναμενόμενο, καθώς ένας λύτης εμπλέκεται με βαριές υπολογιστικές πράξεις, και οι γλώσσες και τα πλαίσια χαμηλότερου επιπέδου είναι συνήθως πιο κατάλληλα για τη διαχείριση των πόρων και της μνήμης του υπολογιστή.

Το Flutter, ως πλαίσιο κυρίως για κινητές εφαρμογές, δεν έχει χρησιμοποιηθεί ευρέως για λύσεις τέτοιου είδους. Ωστόσο, μέσα από την προσέγγιση που ακολουθείται στην παρούσα εργασία, γίνεται μια προσπάθεια να αξιοποιηθεί το Flutter για την ανάπτυξη ενός λύτη, δοκιμάζοντας έτσι τις δυνατότητές του για τέτοιες χρήσεις.

Η δημιουργία ενός λύτη στο Flutter αποτελεί μια καινοτόμα προσέγγιση, με στόχο να εξεταστεί η απόδοση του πλαισίου σε τέτοιες απαιτητικές εφαρμογές και να αναδειχθούν οι δυνατότητες και οι περιορισμοί του.

Κεφάλαιο 6ο: Προετοιμασία Εργασίας

6.1 Εισαγωγή

Στο παρόν κεφάλαιο, εξετάζεται και αναλύεται με λεπτομέρεια όλη η μεθοδολογία που ακολουθήθηκε για την δημιουργία ενός λύτη εικονόσταυρου με τη χρήση Flutter. Γίνεται εκτενής περιγραφή των βημάτων, από τη σύλληψη της ιδέας, τις τεχνολογίες που χρησιμοποιήθηκαν, τις απαιτήσεις, αλλά και τις προκλήσεις και δυσκολίες που έπρεπε να αντιμετωπιστούν μέχρι και το τελικό αποτέλεσμα, τη δημιουργία του λύτη.

Πιο συγκεκριμένα, περιγράφεται η διαδικασία ανάπτυξης της εφαρμογής στο περιβάλλον του Flutter framework και παρουσιάζονται οι βασικές αρχές που καθοδήγησαν τη σχεδίαση και την υλοποίησή της. Επίσης, γίνεται αναφορά στις προκλήσεις που παρουσιάστηκαν κατά τη διάρκεια της ανάπτυξης της εφαρμογής και στις στρατηγικές που ακολουθήθηκαν για την αντιμετώπισή τους. Τέλος, παρουσιάζεται με λεπτομέρεια η προσέγγιση που ακολουθήθηκε για την επίλυση των παζλ Nonogram μέσω του αναπτυγμένου λύτη, συμπεριλαμβανομένων των αλγορίθμων και των ευρημάτων που προέκυψαν.

6.2 Μελέτη Θέματος

6.2.1 Εισαγωγή

Η ενότητα 6.2, με τίτλο Μελέτη Θέματος, αποτελεί το θεμέλιο της πτυχιακής εργασίας, εισάγοντας τη θεωρητική και πρακτική προσέγγιση που ακολουθήθηκε για την ανάπτυξη της εφαρμογής λύσης εικονόσταυρων. Η ενότητα αυτή εξετάζει πώς συνδυάστηκαν οι προσωπικές εμπειρίες, οι ανάγκες της αγοράς και οι τεχνολογικές δυνατότητες για τη σύλληψη και την υλοποίηση ενός έργου που ανταποκρίνεται σε πολλαπλά επίπεδα.

Η υποενότητα 6.2.1 Εισαγωγή έχει ως στόχο να παρουσιάσει το πλαίσιο και τη λογική πίσω από τη δημιουργία του έργου, εξηγώντας πώς το προσωπικό ενδιαφέρον και οι τεχνολογικές ευκαιρίες οδήγησαν στην επιλογή του συγκεκριμένου θέματος. Μέσα από την ανάλυση, αναδεικνύεται ο τρόπος με τον οποίο η δημιουργία μιας εφαρμογής λύσης εικονόσταυρων έγινε όχι μόνο εφικτή αλλά και αναγκαία για την αντιμετώπιση των περιορισμών που παρατηρήθηκαν στις ήδη υπάρχουσες λύσεις.

Η μελέτη ξεκινά με τη σύλληψη της ιδέας, που βασίστηκε στις προσωπικές αναμνήσεις ενασχόλησης με τα εικονόσταυρα και στην επιθυμία να ενσωματωθούν σύγχρονες δυνατότητες σε μια εφαρμογή λύσης παζλ. Στη συνέχεια, μέσω της συλλογής απαιτήσεων, καταγράφονται οι ελλείψεις της αγοράς, όπως η ανάγκη για προσβασιμότητα χωρίς εγκατάσταση, η παροχή διαδραστικών εργαλείων όπως ο συρόμενος ελεγκτής (slider), η υποστήριξη επεξηγήσεων για εκπαιδευτικούς σκοπούς και η επιλογή του Flutter για την ανάπτυξη της εφαρμογής.

Το παρόν κεφάλαιο χρησιμεύει επίσης για την εισαγωγή μιας μεθοδολογίας που διασφαλίζει τη συνέπεια, την απόδοση και την ευχρηστία της εφαρμογής. Παρουσιάζει τα αρχικά βήματα που οδήγησαν στη διαμόρφωση του έργου, ενώ θέτει τη βάση για τις τεχνικές και τις λειτουργικές λεπτομέρειες που ακολουθούν στις επόμενες ενότητες.

Η ενότητα αυτή δεν είναι μόνο θεωρητική, αλλά αποτελεί και έναν διάλογο μεταξύ των προσωπικών εμπειριών και των τεχνολογικών εργαλείων, αναδεικνύοντας την αξία της καινοτομίας στην ανάπτυξη εφαρμογών. Μέσα από αυτή τη διαδικασία, δημιουργήθηκε ένας λύτης εικονόσταυρων που ξεχωρίζει τόσο για τη λειτουργικότητα όσο και για τη φιλικότητα προς τον χρήστη, αποτελώντας ένα σημαντικό βήμα για την εξέλιξη του συγκεκριμένου πεδίου.

6.2.2 Σύλληψη Ιδέας

Από μικρή ηλικία, τα εικονόσταυρα αποτελούσαν ένα κομμάτι διασκέδασης στη ζωή μου. Εγώ και ο αδερφός μου λύναμε τέτοια παζλ στον ελεύθερό μας χρόνο. Η ενασχόλησή μου με αυτά και η εύρεση τρόπων επίλυσής τους ήταν κάτι που με ενδιέφερε και μου κινούσε ιδιαίτερα το ενδιαφέρον από τότε. Με την πάροδο του χρόνου, αρκετές ιδέες που σχετίζονται με τα εικονόσταυρα έχουν περάσει από το μυαλό μου, κάποιες από τις οποίες θα αναλυθούν και παρακάτω ως προσωπικά έργα.

Η πτυχιακή αυτή εργασία, με ενδιέφερε να συνδυάζει την παρουσίαση μιας εφαρμογής και τη βιβλιογραφική ανάλυση ενός θέματος. Με αφορμή αυτό, γεννήθηκε η ιδέα να πραγματευτώ τη δημιουργία μιας εφαρμογής λύτη εικονόσταυρων. Η επιλογή του λύτη ως θέμα της εργασίας έγινε επειδή είναι απαραίτητος για όλα τα έργα που έχω στο μυαλό μου.

6.2.3 Συλλογή Απαιτήσεων

6.2.3.1 Εισαγωγή

Στο προηγούμενο κεφάλαιο αναλύθηκαν οι υπάρχουσες λύσεις που συλλέχθηκαν για μελέτη. Αυτό το βήμα μας δείχνει τι υπάρχει και τι λείπει από την αγορά, έτσι ώστε να συλλεχθούν οι απαιτήσεις της εφαρμογής της εργασίας. Με βάση την αναζήτηση αυτή, έγινε αποτίμηση των χαρακτηριστικών και συλλογή αυτών που είναι βασικών ενός λύτη. Ύστερα, προστέθηκαν χαρακτηριστικά που έλειπαν από τις ήδη υπάρχουσες λύσεις.

Δεν θέλω να προσφέρω άλλον έναν λύτη όπως οι υπάρχοντες, καθώς αυτό δεν πρόκειται να συνεισφέρει κάπως στο τεχνολογικό περιβάλλον, ούτε να προσφέρει κάτι στο πρόβλημα που υπάρχει. Κατά την αναζήτησή μου βρήκα κάποιους πολύ γρήγορους και αποδοτικούς αλγορίθμους, οπότε η προσοχή μου μεταφέρθηκε στις ελλείψεις προσβασιμότητας και εκτέλεσης μιας τέτοιας εφαρμογής.

6.2.3.2 Καμία Ανάγκη για Εγκατάσταση

Μία από τις κύριες απαιτήσεις που εντοπίστηκαν είναι η δυνατότητα χρήσης του λύτη χωρίς την ανάγκη για εγκατάσταση ή ρυθμίσεις. Ο χρήστης πρέπει να μπορεί να εκτελέσει και να χρησιμοποιήσει πλήρως τον λύτη μέσω ενός φυλλομετρητή (browser). Αυτή η προσέγγιση διασφαλίζει ότι η εφαρμογή θα είναι άμεσα διαθέσιμη και εύχρηστη, ανεξάρτητα από το λειτουργικό σύστημα ή τη συσκευή που χρησιμοποιείται.

6.2.3.3 Συρόμενος Ελεγκτής (Slider)

Επιπλέον, αποφασίστηκε η ενσωμάτωση ενός συρόμενου ελεγκτή (slider) που θα επιτρέπει στους χρήστες να κινούνται μπρος και πίσω στα βήματα της λύσης κατά απαίτηση. Αυτή η δυνατότητα παρέχει μεγαλύτερο έλεγχο στον χρήστη, επιτρέποντάς του να μελετήσει τη λύση με τον δικό του ρυθμό.

6.2.3.4 Επεξηγήσεις Βημάτων

Ένα άλλο κρίσιμο χαρακτηριστικό που προστέθηκε είναι η παροχή επεξηγήσεων για κάθε βήμα της λύσης, όποτε αυτό είναι δυνατόν. Οι επεξηγήσεις αυτές βοηθούν τους χρήστες να κατανοήσουν όχι μόνο το τι συμβαίνει, αλλά και το γιατί.

6.2.3.5 Χρήση Flutter

Τέλος, αποφασίστηκε η υλοποίηση όλων των παραπάνω χρησιμοποιώντας το framework Flutter. Η επιλογή του Flutter έγινε λόγω της ευελιξίας και της αποδοτικότητας που προσφέρει στην ανάπτυξη εφαρμογών, ιδιαίτερα για κινητές συσκευές.

6.2.4 Σύνοψη Συλλογής Απαιτήσεων

Συνοψίζοντας, οι απαιτήσεις για την ανάπτυξη του λύτη περιλαμβάνουν βασικά χαρακτηριστικά που εξασφαλίζουν ευχρηστία, κατανόηση και αποδοτικότητα. Αυτές περιλαμβάνουν:

1. **Καμία ανάγκη για εγκατάσταση:** Ο λύτης να μπορεί να εκτελείται πλήρως μέσω ενός φυλλομετρητή, χωρίς να απαιτείται οποιαδήποτε εγκατάσταση ή ρύθμιση.
2. **Συρόμενος ελεγκτής (slider):** Ενσωμάτωση ενός συρόμενου ελεγκτή που επιτρέπει στους χρήστες να κινούνται μπρος και πίσω στα βήματα της λύσης κατά απαίτηση.
3. **Επεξηγήσεις βημάτων:** Παροχή επεξηγήσεων για κάθε βήμα της λύσης, όποτε αυτό είναι δυνατόν, για να βοηθήσει τους χρήστες να κατανοήσουν τη λογική πίσω από κάθε κίνηση.
4. **Χρήση Flutter:** Υλοποίηση όλων των παραπάνω χρησιμοποιώντας το framework Flutter, για ευελιξία και αποδοτικότητα στην ανάπτυξη εφαρμογών.

Επιπλέον, υπάρχουν και απαιτήσεις ευρύτερων στόχων (*stretch goals*), οι οποίες θα αναλυθούν σε επόμενη ενότητα της εργασίας. Αυτές οι απαιτήσεις αφορούν δυνατότητες βελτίωσης και προτάσεις αναβάθμισης της εφαρμογής, διασφαλίζοντας τη συνεχή εξέλιξη και καινοτομία του έργου.

6.3 Οργάνωση Εργασιών

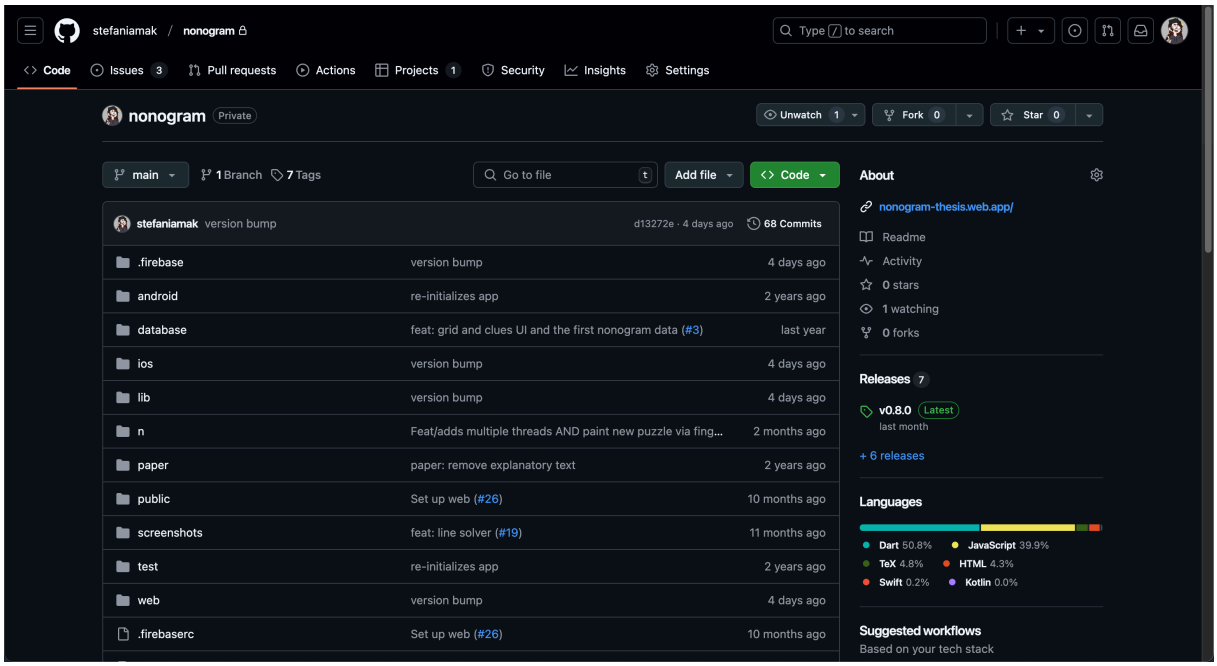
6.3.1 Εισαγωγή

Για τη διαχείριση και οργάνωση της ανάπτυξης της εφαρμογής, δημιουργήθηκε ένα αποθετήριο (repository) στον ιστότοπο του GitHub. Μέσω του GitHub Projects και του συστήματος Kanban, επιτεύχθηκε η αποτελεσματική παρακολούθηση της προόδου των εργασιών.

6.3.2 GitHub

Το GitHub χρησιμοποιήθηκε για την αποθήκευση και διαμοίραση του κώδικα της εφαρμογής, ενώ προσέφερε πρόσθετες δυνατότητες διαχείρισης, όπως το σύστημα GitHub Projects. Η επιλογή του GitHub έγινε λόγω της ευρείας αποδοχής του στον τομέα, της δωρεάν χρήσης του για ατομικές ανάγκες, και της υποστήριξης του ανοικτού κώδικα.

Στο πλαίσιο αυτό, δημιουργήθηκε ένα ιδιωτικό αποθετήριο, με την προοπτική να γίνει δημόσιο μετά την ολοκλήρωση της πτυχιακής εργασίας. Το αποθετήριο περιλαμβάνει αρχεία κώδικα, τεκμηρίωσης και άλλα βοηθητικά στοιχεία για τη διαχείριση του έργου.



Σχήμα 6.1: Το αποθετήριο της εφαρμογής στο GitHub

6.3.3 GitHub Projects

Μια αρκετά νέα λειτουργία του GitHub είναι το GitHub Projects. Αυτή η δυνατότητα αφορά την καταγραφή και οργάνωση των θεμάτων (issues) και των εργασιών (tasks), καθώς και τη σύνδεσή τους με τον κώδικα ή άλλες πληροφορίες. Προσφέρει μια οργανωμένη καταγραφή του τι έχει γίνει και τι πρέπει να γίνει, μαζί με τα ευρήματα και τα αποτελέσματα, διευκολύνοντας έτσι τη διαχείριση του έργου και την παρακολούθηση της προόδου του.

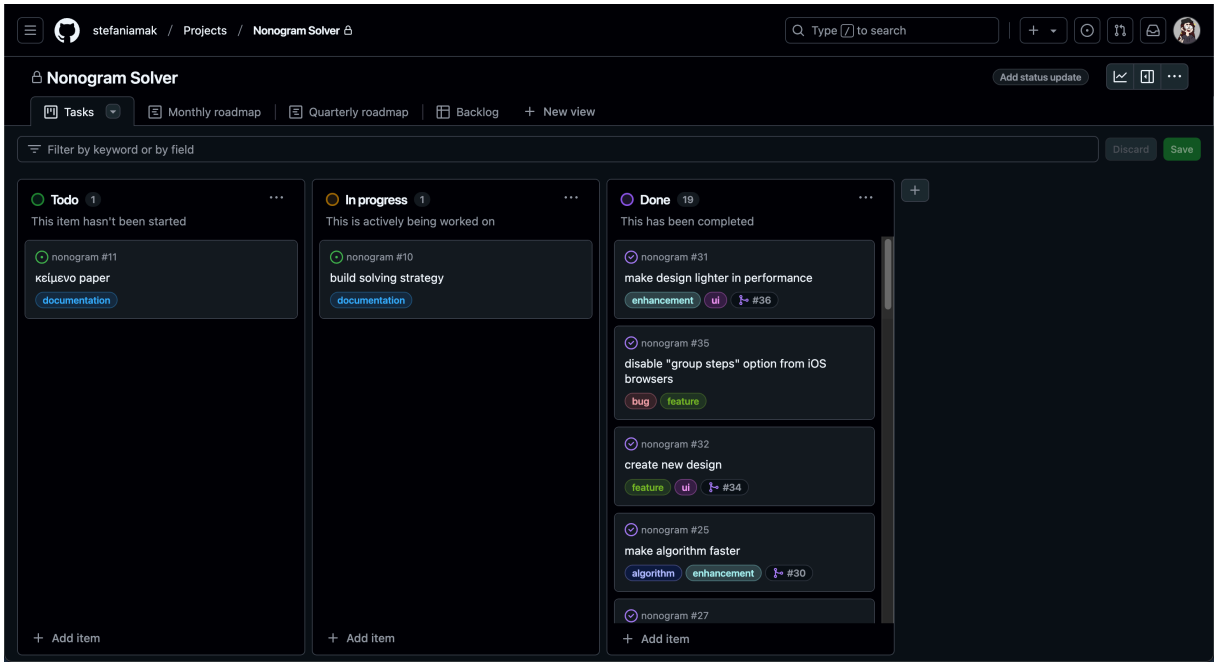
Στο πλαίσιο αυτό, δημιουργήθηκε ένα project πάνω στο repository της εργασίας, με ένα κενό Kanban τριών στηλών: “Προς Εκτέλεση”, “Σε Εξέλιξη” και “Ολοκληρωμένο”, το οποίο αντιστοιχεί σε “αναμένεται να ξεκινήσει”, “δουλεύεται αυτή τη στιγμή” και “έχει ολοκληρωθεί” αντίστοιχα.

Πιο συνοπτικά, οι τρεις στήλες του Kanban είναι οι εξής:

- **Προς Εκτέλεση:** Αναπαριστά τις εργασίες που έχουν αναγνωριστεί αλλά δεν έχουν ξεκινήσει.
- **Σε Εξέλιξη:** Αναπαριστά τις εργασίες που βρίσκονται υπό υλοποίηση.

- **Ολοκληρωμένο:** Περιλαμβάνει τις εργασίες που έχουν ολοκληρωθεί.

Η συμπλήρωση των στηλών με έργα ξεκίνησε από το στάδιο της μελέτης ήδη υπαρχόντων αλγορίθμων. Έτσι, δημιουργήθηκαν εργασίες με σκοπό την αναζήτηση στοχευμένων πληροφοριών. Αυτές οι εργασίες αφορούσαν είτε τη μελέτη συγκεκριμένων αλγορίθμων είτε την έρευνα για κοινά χαρακτηριστικά τους (όπως ο τρόπος που εισάγουν παζλ στα προγράμματά τους) είτε τον σχεδιασμό των λειτουργιών και τεχνικών που θα εφαρμοστούν στο τρέχον έργο.



Σχήμα 6.2: Το GitHub Projects της εργασίας

6.3.4 Ροή Εργασίας

6.3.4.1 Εισαγωγή

Για να εξασφαλιστεί η ομαλή ροή εργασίας και η αποτελεσματική παρακολούθηση της προόδου, ακολουθήθηκε μια σειρά από καλά καθορισμένες πρακτικές. Οι πρακτικές αυτές σχεδιάστηκαν για να διευκολύνουν την ανάπτυξη της εφαρμογής, να μειώσουν τα σφάλματα και να επιτρέψουν τη διατήρηση ενός σαφούς ιστορικού αλλαγών.

Οι βασικοί στόχοι της υιοθέτησης αυτών των πρακτικών περιλάμβαναν:

- Τη διαχείριση νέων λειτουργιών και διορθώσεων με έναν τρόπο που να ελαχιστοποιεί τον κίνδυνο εισαγωγής νέων σφαλμάτων.
- Τη διατήρηση της καθαρότητας του κώδικα, με οργανωμένο ιστορικό αλλαγών.
- Τη διευκόλυνση της συνεργασίας και της αναθεώρησης του κώδικα μέσω μιας κεντροποιημένης πλατφόρμας.
- Την εξασφάλιση ότι κάθε αλλαγή τεκμηριώνεται σωστά και μπορεί να ανιχνευθεί στο μέλλον.

Οι δύο βασικές διαδικασίες που χρησιμοποιήθηκαν για την επίτευξη αυτών των στόχων ήταν η δημιουργία Pull Requests (PRs) και η διαχείριση εκδόσεων.

6.3.4.2 Δημιουργία Pull Request για κάθε νέο χαρακτηριστικό ή διόρθωση

Η οργάνωση της ανάπτυξης βασίστηκε στη δημιουργία ενός Pull Request (PR) για κάθε νέο χαρακτηριστικό ή διόρθωση σφαλμάτων. Κάθε PR σχετιζόταν με ένα ξεχωριστό branch, το οποίο δημιουργούνταν αποκλειστικά για το συγκεκριμένο χαρακτηριστικό ή τη διόρθωση. Αυτή η πρακτική διαχωρισμού των εργασιών βοήθησε στη διατήρηση της καθαρότητας του κύριου branch, αποτρέποντας την ανάμειξη ημιτελών αλλαγών με την πλήρη εφαρμογή.

Πλεονεκτήματα της προσέγγισης:

- Διευκόλυνε την οργάνωση, υπήρχε μια καθαρή εικόνα του τι είχε ολοκληρωθεί και τι έχει μπει πότε στο τελικό πρότζεκτ.
- Παρείχε ένα σαφές και ανιχνεύσιμο ιστορικό αλλαγών, διευκολύνοντας την κατανόηση των προσθηκών και τροποποιήσεων.
- Μείωσε τον κίνδυνο σύγκρουσης αλλαγών (merge conflicts), αφού τα PRs συγχωνεύονταν μόνο αφού είχαν αναθεωρηθεί και δοκιμαστεί πλήρως.

Κάθε PR περιλάμβανε λεπτομερή περιγραφή της εργασίας που ολοκληρώθηκε, με σχόλια και σημειώσεις για την κατανόηση των αλλαγών. Οι πλατφόρμες σχολιασμού του GitHub χρησιμοποιήθηκαν για την προσθήκη παρατηρήσεων, διευκρινίσεων ή προτάσεων βελτίωσης, διασφαλίζοντας ότι ο κώδικας ήταν πλήρως κατανοητός πριν από τη συγχώνευση.

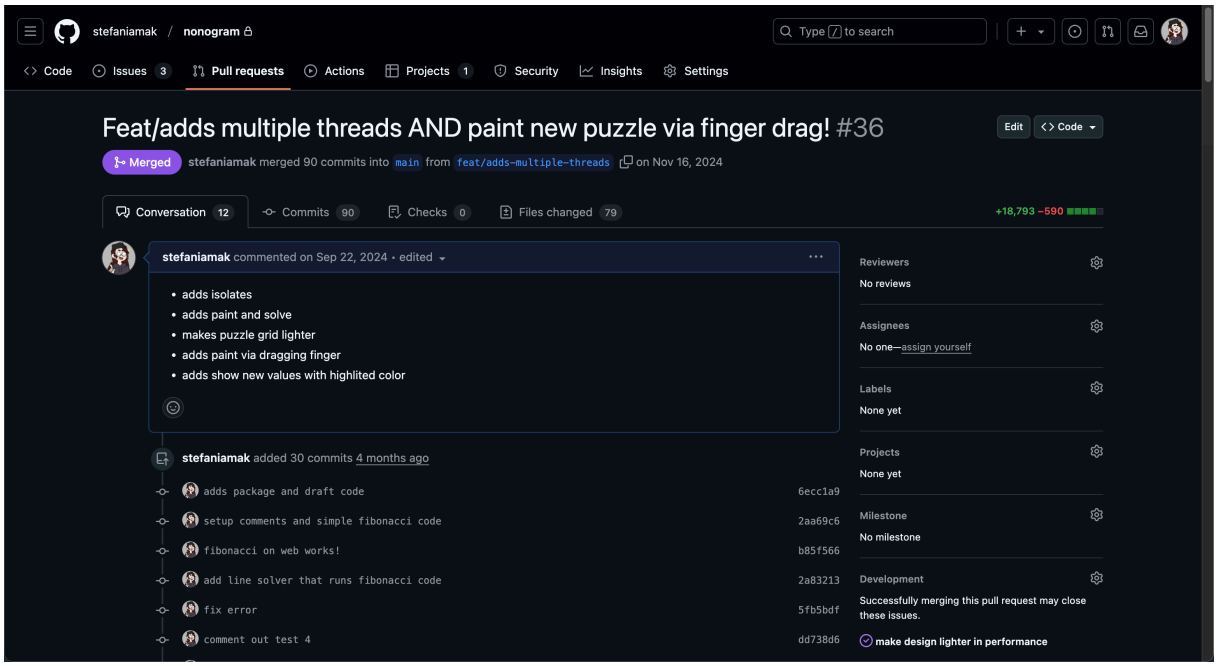
Μετά την ολοκλήρωση του PR:

- Ο κώδικας αναθεωρούνταν διεξοδικά για να διασφαλιστεί ότι πληρούσε τα απαιτούμενα κριτήρια ποιότητας και λειτουργικότητας.
- Το PR συγχωνευόταν (*merged*) στο κύριο branch.
- Το σχετικό branch διαγραφόταν, διατηρώντας έτσι το αποθετήριο καθαρό και οργανωμένο.

Αυτή η προσέγγιση διατήρησε το ιστορικό του κύριου branch καθαρό και οργανωμένο, ενώ εξασφάλισε ότι κάθε αλλαγή ήταν ανιχνεύσιμη και μπορούσε να αναθεωρηθεί με ακρίβεια.

Παραδείγματα χρήσης PR:

- Ανάπτυξη νέου χαρακτηριστικού: Ένα νέο feature για τη δυνατότητα επεξήγησης βημάτων λύσης δημιουργήθηκε σε ξεχωριστό branch και καταγράφηκε λεπτομερώς στο PR.
- Διόρθωση σφαλμάτων: Μια αλλαγή που αφορούσε τη βελτίωση της διαχείρισης μνήμης έγινε μέσω ξεχωριστού PR, επιτρέποντας την αξιολόγηση του αντικτύπου της πριν την ενσωμάτωση.



Σχήμα 6.3: Διαδικασία δημιουργίας και συγχώνευσης Pull Request στο GitHub.

- Ενημερώσεις ντοκουμέντων: Ακόμη και αλλαγές στα αρχεία τεκμηρίωσης της εφαρμογής πραγματοποιήθηκαν μέσω PR, εξασφαλίζοντας ότι οι περιγραφές παρέμεναν ενημερωμένες και ακριβείς.

Η εφαρμογή αυτών των πρακτικών διασφάλισε την ποιότητα και τη διαφάνεια κατά την ανάπτυξη της εφαρμογής, ενώ παράλληλα διευκόλυνε τη συνεργασία και τη διατήρηση ενός οργανωμένου και καθαρού αποθετηρίου.

6.3.4.3 Έκδοση εκδόσεων (Versioning)

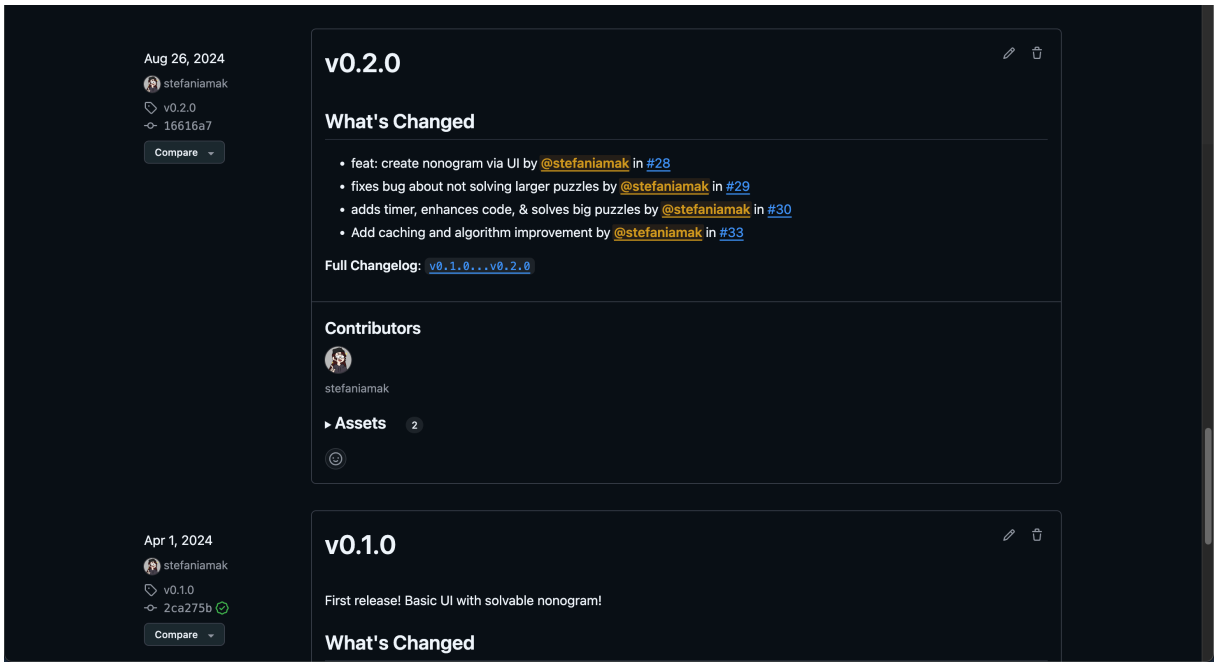
Η διαχείριση των εκδόσεων της εφαρμογής αποτέλεσε σημαντικό κομμάτι της ροής εργασίας. Ακολουθήθηκε μια συγκεκριμένη στρατηγική αρίθμησης εκδόσεων, η οποία αποτελείτο από τρία επίπεδα αριθμών (Μείζων.Δευτερεύων.Ελάσσων):

- **Μείζων αριθμός:** Αντιπροσωπεύει την τελική έκδοση της εφαρμογής. Για παράδειγμα, το v1.0.0 αναφέρεται στην έκδοση που παραδόθηκε ως τελικό παραδοτέο της πτυχιακής. Όποια έκδοση έχει αριθμό μεγαλύτερο του v1.0.0 υποδεικνύει αλλαγές ή βελτιώσεις που έγιναν μετά την παράδοση της εργασίας.
- **Δευτερεύων αριθμός:** Δευτερεύων αριθμός: Αντιπροσωπεύει την προσθήκη μεγάλων νέων χαρακτηριστικών. Συνήθως ενημερώνεται κάθε φορά που ολοκληρώνεται ένα PR και συγχωνεύεται στο κύριο branch.
- **Ελάσσων αριθμός:** Ελάσσων αριθμός: Υποδεικνύει μικρές αλλαγές, όπως διορθώσεις ή τροποποιήσεις που δεν επηρεάζουν σημαντικά τη λειτουργικότητα. Αυτός ο αριθμός αυξάνεται με κάθε νέο build που ανεβαίνει, διευκολύνοντας την επιβεβαίωση πως η ενημέρωση της εφαρμογής έγινε σωστά.

Για να διατηρηθεί ένα καθαρό χρονικό αρχείο των αλλαγών που έγιναν σε κάθε έκδοση, χρησιμοποιήθηκε το τμήμα Releases του GitHub. Σε αυτό το τμήμα δημιουργήθηκαν ετικέτες (tags) που αντιπροσωπεύουν τις εκδόσεις της εφαρμογής. Οι περιγραφές των εκδόσεων περιλάμβαναν αυτόματα τους τίτλους των PRs που συγχωνεύθηκαν από την προηγούμενη έκδοση έως τη νέα.

Ακολουθώντας την πρακτική δημιουργίας ενός PR για κάθε νέο χαρακτηριστικό ή διόρθωση, η περιγραφή των νέων λειτουργιών που προστέθηκαν σε κάθε έκδοση δημιουργήθηκε εύκολα και με ακρίβεια. Για λόγους σαφήνειας και ευαναγνωσίας, τα Releases δημιουργούνταν μόνο όταν υπήρχε αλλαγή στον μείζονα ή δευτερεύοντα αριθμό της έκδοσης, όχι στον ελάσσονα.

Με αυτές τις πρακτικές, εξασφαλίστηκε η διαφάνεια και η οργανωμένη καταγραφή των αλλαγών, επιτρέποντας την αποτελεσματική διαχείριση της ανάπτυξης της εφαρμογής.



Σχήμα 6.4: Η σελίδα των εκδόσεων στο GitHub

Κεφάλαιο 7ο: Σχεδιασμός

7.1 Εισαγωγή

Η διαδικασία σχεδιασμού της εφαρμογής ήταν μια δημιουργική και ταυτόχρονα συστηματική προσπάθεια που ξεκίνησε από την ανάπτυξη προσχεδίων σε χαρτί και κατέληξε στη δημιουργία ψηφιακών πρωτοτύπων υψηλής πιστότητας. Σε αυτή την ενότητα, περιγράφεται πώς οι αρχικές ιδέες και προσεγγίσεις εξελίχθηκαν σε συγκεκριμένα σχέδια που καθοδήγησαν την υλοποίηση.

Η διαδικασία σχεδιασμού ξεκίνησε με τη χρήση παραδοσιακών εργαλείων, όπως το χαρτί και το μολύβι, τα οποία αποτέλεσαν το πρώτο βήμα για την κατανόηση και τον ορισμό της συνολικής εμπειρίας χρήστη. Η ελευθερία που παρέχουν αυτά τα εργαλεία διευκόλυνε τη δημιουργική σκέψη και την ανάλυση πολλών διαφορετικών ιδεών.

Στη συνέχεια, οι ιδέες μεταφέρθηκαν στο *Figma*, ένα σύγχρονο και ισχυρό εργαλείο σχεδιασμού που χρησιμοποιείται ευρέως για τη δημιουργία διαδραστικών πρωτοτύπων. Με τη βοήθεια του *Figma*, τα σχέδια εξελίχθηκαν σε λεπτομερή μοντέλα που περιλάμβαναν διαδραστικά στοιχεία, μεταβάσεις μεταξύ σελίδων και συνεπή οπτική ταυτότητα.

Η συνδυαστική χρήση παραδοσιακών και ψηφιακών εργαλείων διασφάλισε ότι το τελικό προϊόν ήταν τόσο λειτουργικό όσο και φιλικό προς τον χρήστη, ενώ παράλληλα η διαδικασία σχεδιασμού παραμένει *ευέλικτη* και **προσαρμόσιμη** σε διαφορετικές απαιτήσεις.

7.2 Χαρτί και Μολύβι

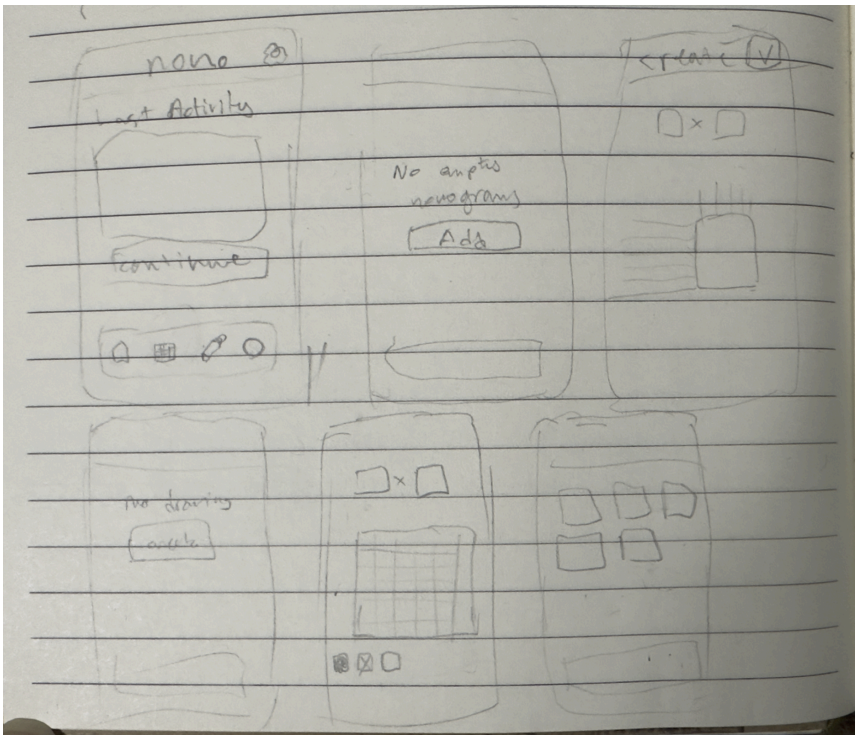
Ο σχεδιασμός των σελίδων ξεκίνησε με τη δημιουργία προσχεδίων στο χαρτί, όπου έγινε η πρώτη εκτίμηση για το περιεχόμενο και τις ανάγκες κάθε σελίδας, δηλαδή την **Εμπειρία Χρήστη (User Experience)**. Η διαδικασία αυτή επέτρεψε την ελεύθερη ροή ιδεών και την εξέταση διαφορετικών προσεγγίσεων πριν την ψηφιακή τους υλοποίηση.

Τα αρχικά σχέδια στο χαρτί περιλάμβαναν σκίτσα των βασικών διατάξεων, τη θέση των κουμπιών και άλλων στοιχείων διεπαφής, καθώς και τη δομή της πλοήγησης μεταξύ των σελίδων. Αυτή η προσέγγιση έδωσε τη δυνατότητα για **γρήγορες αλλαγές** και *διορθώσεις*, κάτι που θα ήταν πιο χρονοβόρο σε ψηφιακή μορφή. Η ευελιξία του χαρτιού και του μολυβιού επέτρεψε την εύκολη τροποποίηση των σχεδίων, τη μετακίνηση στοιχείων και την προσαρμογή της διάταξης ανάλογα με τις απαιτήσεις που προέκυπταν.

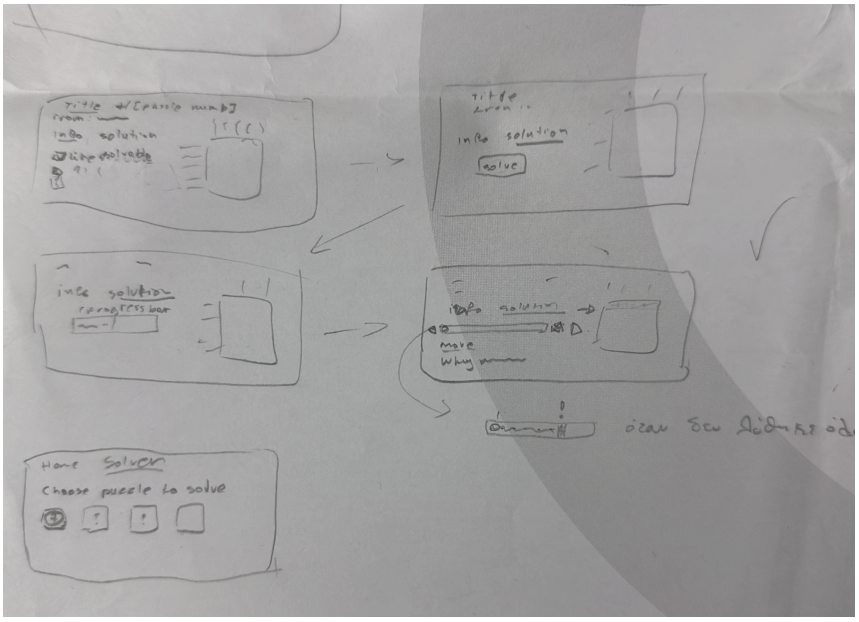
Παρά το γεγονός ότι το τελικό αποτέλεσμα υλοποιείται ψηφιακά, το χαρτί και το μολύβι θεωρούνται **ιδανικά** για το αρχικό *«brainstorming»*. Αυτό επιτρέπει τη δημιουργία προσχεδίων για το πώς θα τοποθετηθούν τα επιμέρους στοιχεία χωρίς να χρειάζεται οριστική δέσμευση για τη μορφή ή το μέγεθός τους. Πρόκειται, ουσιαστικά, για μια «πρόχειρη» μα ωστόσο πολύ *δημιουργική* φάση, όπου τα λάθη είναι εύκολο να διορθωθούν και οι επιλογές παραμένουν ανοιχτές.

Σχήματα 7.1 και 7.2 απεικονίζουν τα αρχικά σχέδια σε χαρτί, τόσο για την έκδοση της εφαρμογής για κινητές συσκευές όσο και για την ιστοσελίδα. Αυτές οι εικόνες αναδεικνύουν τη δημιουργική διαδικασία, αποτυπώνοντας τη σκέψη πίσω από τον σχεδιασμό.

Η χρήση χαρτιού και μολυβιού σε αυτό το σημείο προσφέρει την ελευθερία να κάνουμε πολλά λάθη και να μεταφέρουμε εύκολα τις ιδέες μας από το ένα σχέδιο στο άλλο. Αυτό επιτρέπει στη *σκέψη* και τη *φαντασία* να εξελιχθούν και να εμπνευστούν χωρίς περιορισμούς, δημιουργώντας μια ισχυρή βάση για τον ψηφιακό σχεδιασμό που θα ακολουθήσει.



Σχήμα 7.1: Αρχικά σχέδια εφαρμογής κινητού



Σχήμα 7.2: Αρχικά σχέδια εφαρμογής ιστοσελίδας

Η χρήση παραδοσιακών εργαλείων αποτέλεσε τη βάση για τον ψηφιακό σχεδιασμό που ακολούθησε, δημιουργώντας ένα *ισχυρό θεμέλιο* για τη συνολική εμπειρία χρήστη.

7.3 Χρήση Figma

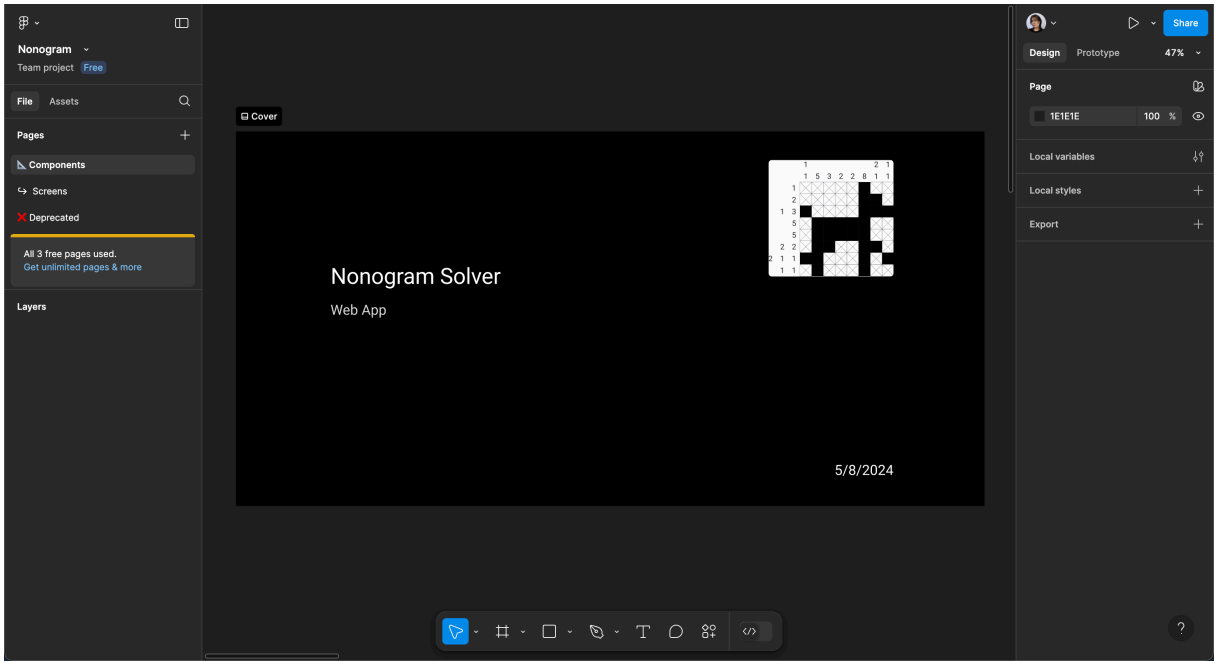
7.3.1 Εισαγωγή

Το *Figma* έχει καθιερωθεί ως ένα από τα **κύρια εργαλεία σχεδιασμού** που χρησιμοποιούνται από πολλές εταιρείες για τη δημιουργία ιστοσελίδων και εφαρμογών. Η **ευκολία χρήσης** του, καθώς είναι προσβάσιμο εξ ολοκλήρου διαδικτυακά, το καθιστά *ασυναγώνιστο*. Η δυνατότητα πρόσβασης στο εργαλείο **ανά πάσα στιγμή** και από *οποιαδήποτε συσκευή* επιτρέπει στους σχεδιαστές να εργάζονται **αποδοτικά** και χωρίς περιορισμούς.

Αφού ολοκληρώθηκαν τα προσχέδια στο χαρτί, η προσοχή στράφηκε στη **μεταφορά αυτών των ιδεών** στο ψηφιακό περιβάλλον, χρησιμοποιώντας το πρόγραμμα *Figma*. Το *Figma* διευκόλυνε την υλοποίηση των προσχεδίων σε **υψηλής πιστότητας πρωτότυπα**, επιτρέποντας την **ακριβή απεικόνιση** των στοιχείων διεπαφής και την *προσομοίωση* της πλοήγησης μεταξύ των σελίδων. Επιπλέον, το *Figma* παρείχε τη δυνατότητα **συνεργασίας σε πραγματικό χρόνο**, διευκολύνοντας την ανταλλαγή απόψεων και την *κοινή εργασία* με την ομάδα ανάπτυξης.

Η διαδικασία του ψηφιακού σχεδιασμού περιλάμβανε επίσης τη **δοκιμή διαφορετικών χρωματικών συνδυασμών**, *τυπογραφικών επιλογών* και *εικονιδίων*, με στόχο τη δημιουργία μιας **συνεκτικής** και *ευχάριστης εμπειρίας χρήστη*. Ιδιαίτερη έμφαση δόθηκε στη **φιλικότητα προς τον χρήστη** και την *προσβασιμότητα*, εξασφαλίζοντας ότι η τελική εφαρμογή θα είναι **εύχρηστη** για *όλους τους χρήστες*.

Ο σχεδιασμός των σελίδων ολοκληρώθηκε με τη δημιουργία ενός **ολοκληρωμένου πρωτοτύπου** στο *Figma*, το οποίο περιλάμβανε όλες τις **βασικές σελίδες** της εφαρμογής, τα *διαδραστικά στοιχεία* και τις **μεταβάσεις μεταξύ των σελίδων**. Αυτό το πρωτότυπο χρησίμευσε ως **οδηγός** για την ανάπτυξη της εφαρμογής, διασφαλίζοντας ότι όλες οι λειτουργίες και οι απαιτήσεις σχεδιασμού θα υλοποιηθούν με *ακρίβεια* κατά τη φάση της ανάπτυξης.



Σχήμα 7.3: Αρχική σελίδα εφαρμογής στο Figma.

7.3.2 Δημιουργία Εξαρτημάτων (Components)

7.3.2.1 Τι είναι τα εξαρτήματα

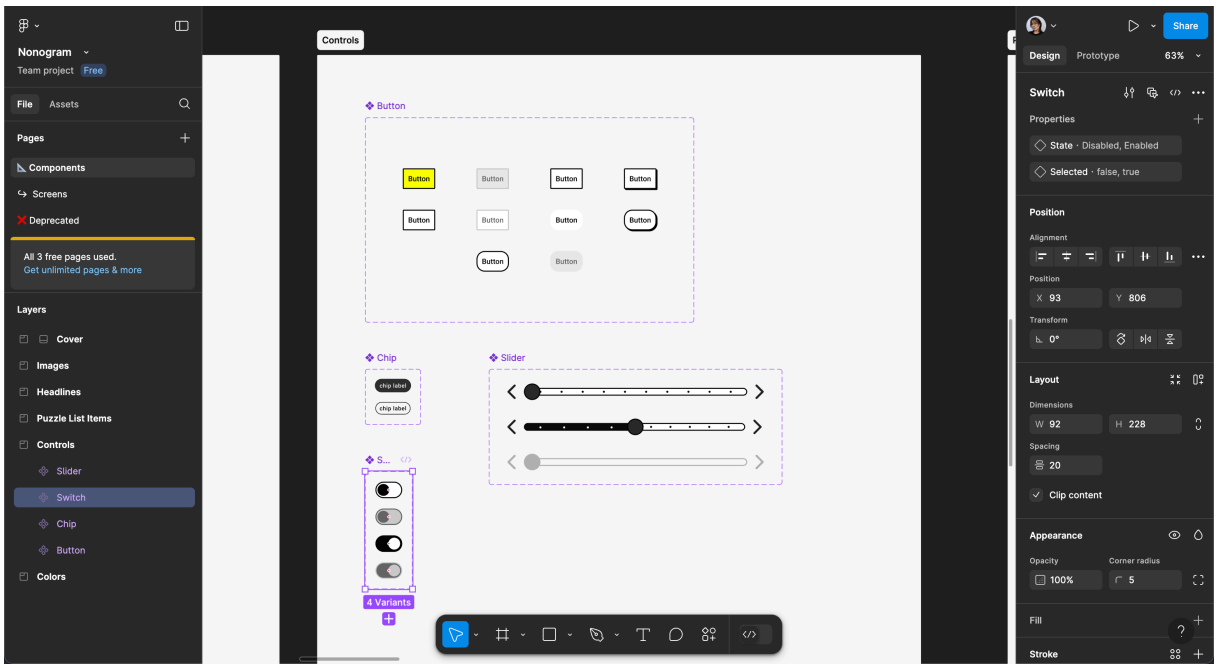
Στο *Figma*, τα εξαρτήματα (*components*) αποτελούν **επαναχρησιμοποιήσιμα γραφιστικά στοιχεία** που μπορούν να εισαχθούν σε πολλές σελίδες ή αρχεία. Η χρησιμότητά τους έγκειται στο ότι επιτρέπουν τη μαζική ενημέρωση όλων των εμφανίσεων ενός στοιχείου μέσω μιας κεντρικής αλλαγής, χωρίς να χρειάζεται μεμονωμένη τροποποίηση σε κάθε σελίδα.

7.3.3 Κύρια Εξαρτήματα (Components)

Το πρώτο βήμα κατά τη χρήση του *Figma* ήταν η δημιουργία των κύριων εξαρτημάτων (*components*) της εφαρμογής. Αυτό περιλάμβανε:

- **Κουμπιά (Buttons):** Κατασκευάστηκαν κουμπιά με διαφορετικές καταστάσεις (*states*), όπως ενεργά, ανενεργά και πατημένα (*hover, disabled, pressed*).
- **Ολισθητές (Sliders):** Ενσωματώθηκαν sliders για την επιλογή βημάτων ή παραμέτρων.
- **Τσιπς (Chips):** Δημιουργήθηκαν μικρά οπτικά στοιχεία που χρησιμοποιούνται για την επιλογή κατηγοριών ή φίλτρων.
- **Διακόπτες (Switches):** Σχεδιάστηκαν διακόπτες με διαφορετικές καταστάσεις, όπως ενεργοποιημένοι και απενεργοποιημένοι.

Για κάθε εξάρτημα δημιουργήθηκαν πολλαπλές εκδοχές που κάλυπταν διαφορετικές αλληλεπιδράσεις του χρήστη, προσφέροντας μια ολοκληρωμένη και συνεπή εμπειρία χρήστη.



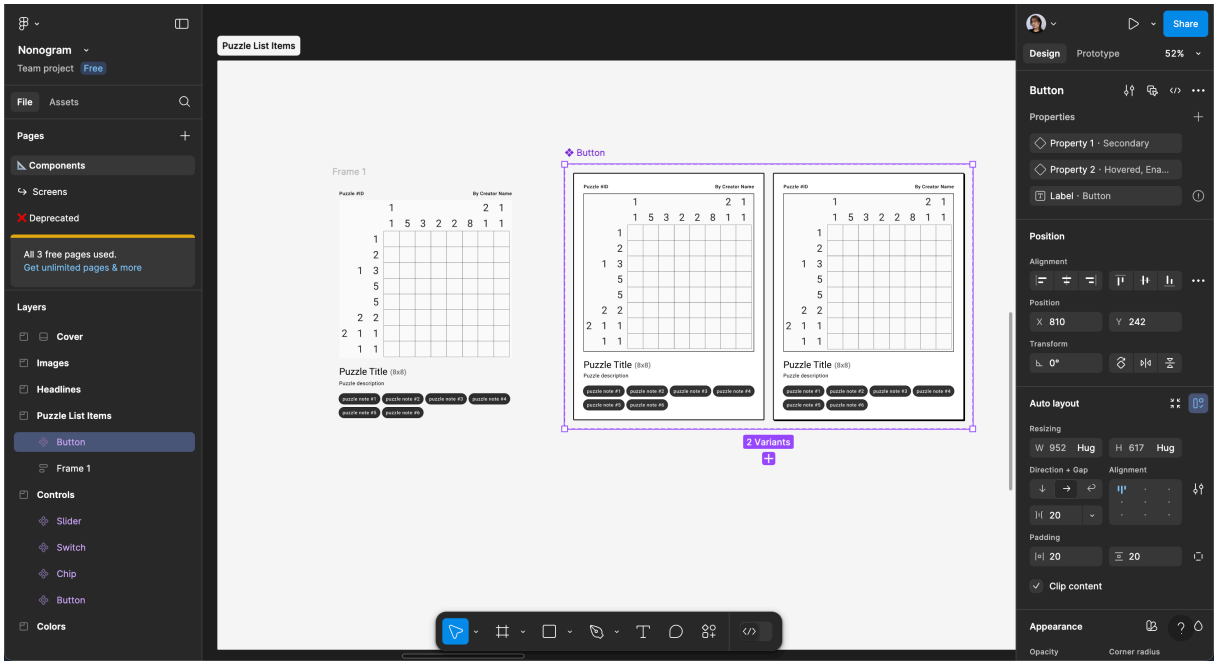
Σχήμα 7.4: Γραφιστικά των κύριων εξαρτημάτων.

7.3.4 Βοηθητικά Εξαρτήματα (Components)

Μετά τη δημιουργία των βασικών εξαρτημάτων, αναπτύχθηκαν πιο εξειδικευμένα εξαρτήματα για την εφαρμογή, όπως:

- **Στοιχεία λίστας (List Items):** Εμφάνιση εικονόσταυρων με εικόνες, περιγραφή και χαρακτηριστικά.
- **Επικεφαλίδες και Υποσέλιδα (Headers και Footers):** Στοιχεία για την οργάνωση των σελίδων.
- **Πρότυπο Εικονόσταυρου:** Μια βασική δομή που περιλάμβανε το πλέγμα του παζλ και πολλαπλές καταστάσεις (*states*).

Αυτά τα εξαρτήματα σχεδιάστηκαν ώστε να είναι επαναχρησιμοποιήσιμα, εξασφαλίζοντας συνέπεια και ταχύτητα στον σχεδιασμό.



Σχήμα 7.5: Βοηθητικά εξαρτήματα ενδεικτικά μαζεμένα.

7.3.5 Δημιουργία Σελίδων

7.3.5.1 Εισαγωγή

Η δημιουργία των σελίδων ξεκίνησε με τη συνδυαστική χρήση των κύριων και βοηθητικών εξαρτημάτων, μαζί με επιπλέον γραφιστικά στοιχεία. Ο στόχος ήταν η ανάπτυξη ενός συνεκτικού σχεδιασμού για τις διαφορετικές εκδόσεις της εφαρμογής (για κινητές συσκευές και υπολογιστές).

7.3.5.2 Λίστα Επιλογής Εικονόσταυρου

Μία από τις πρώτες σελίδες που σχεδιάστηκαν ήταν η λίστα επιλογής εικονόσταυρου. Το σχέδιο προσαρμόστηκε για να υποστηρίζει διαφορετικά μεγέθη οθόνης:

- Σε κινητές συσκευές, η διάταξη είναι κατακόρυφη, επιτρέποντας την εύκολη πλοήγηση.
- Σε μεγαλύτερες οθόνες, τα στοιχεία οργανώνονται σε σειρές, διατηρώντας συγκεκριμένο όριο πλάτους ώστε η λίστα να μην «απλώνεται» υπερβολικά.

Σχήμα 7.6: Σελίδα επιλογής εικονόσταυρου.

7.3.5.3 Σελίδα Αυτόματης Λύσης Εικονόσταυρου

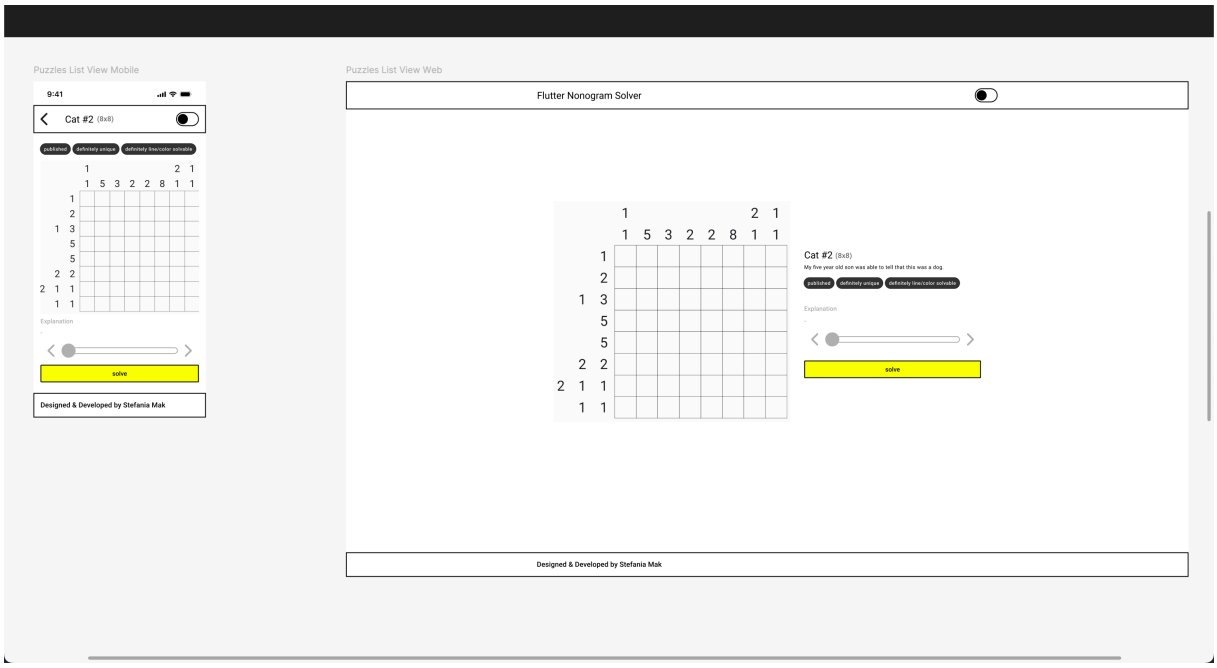
Η σελίδα αυτόματης λύσης περιλαμβάνει δύο κύριες καταστάσεις: μία πριν την εκτέλεση του αλγορίθμου και μία που εμφανίζεται μετά την ολοκλήρωση της επίλυσης. Πιο συγκεκριμένα, αρχικά ο χρήστης βλέπει έναν κενό ή μη συμπληρωμένο εικονόσταυρο, μαζί με το κουμπί **«ΕΠΙΛΥΣΗ»**. Όταν πατηθεί το κουμπί, ο αλγόριθμος εκτελείται και η σελίδα μεταβαίνει αυτόματα στην κατάσταση **«μετά την επίλυση»**, όπου εμφανίζεται το λυμένο εικονόσταυρο.

Στο στάδιο αυτό, έχει προστεθεί ένας ολισθητήρας (*slider*) που επιτρέπει στον χρήστη να επιλέξει ενδιάμεσα βήματα της λύσης. Έτσι, μπορεί να μετακινήσει τον δείκτη του ολισθητήρα και να δει πώς εξελίχθηκε η διαδικασία επίλυσης βήμα προς βήμα, γεγονός που διευκολύνει την κατανόηση του αλγορίθμου αλλά και τη διερεύνηση πιθανών εναλλακτικών.

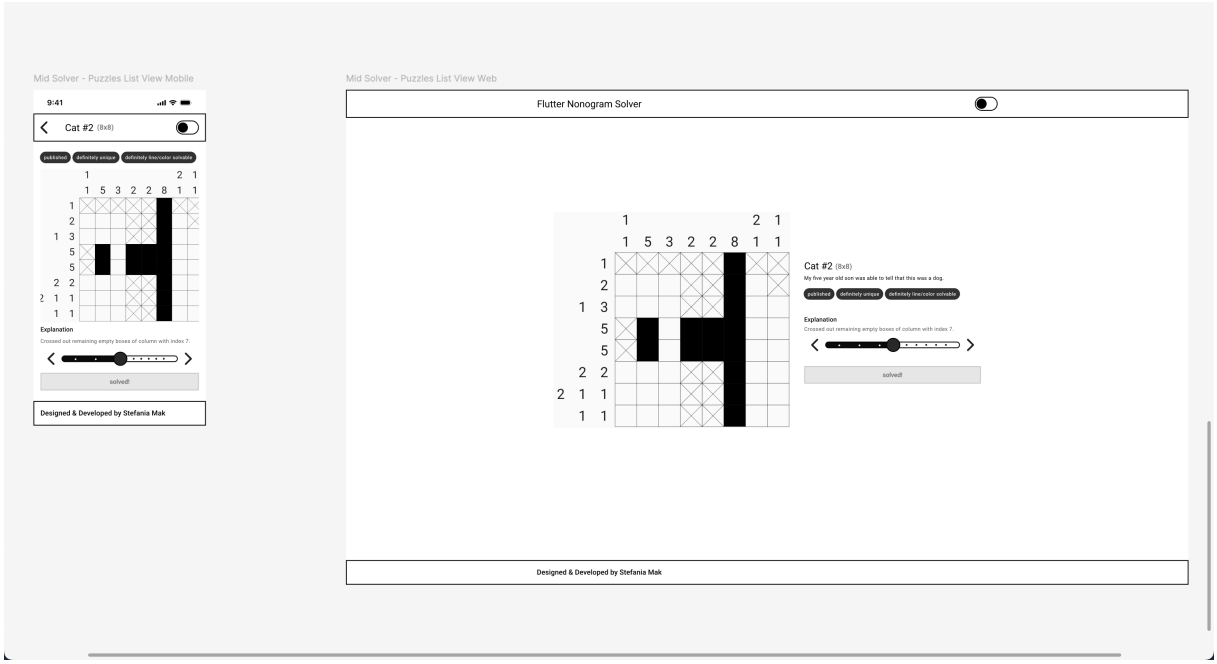
Σχεδιαστικά, η σελίδα περιλαμβάνει ένα πλαίσιο που προβάλλει το πλέγμα του εικονόσταυρου σε ευδιάκριτη μορφή, τα σχετικά στατιστικά ή ενδείξεις λύσης (εφόσον υποστηρίζονται) και το slider στο κάτω ή στο πλάι, ανάλογα με τη διαθέσιμη διάσταση οθόνης. Αυτή η διάταξη επιτρέπει την εύκολη εστίαση στο παζλ, ενώ παράλληλα προσφέρει τις επιπλέον λειτουργίες (παρακολούθηση ενδιάμεσων βημάτων, επιστροφή σε προγενέστερα στάδια κ.λπ.) χωρίς να διαταράσσεται η κυρίως προβολή.

Τα γραφιστικά που χρησιμοποιούνται έχουν προσαρμοστεί σε επιμέρους παραμέτρους, όπως το μέγεθος του πλέγματος ή τα κουμπιά ελέγχου, και εμπλουτίστηκαν σταδιακά καθώς προέκυψαν απαιτήσεις κατά

τη διάρκεια της υλοποίησης. Με αυτόν τον τρόπο, επιτυγχάνεται μια σελίδα που συνδυάζει τη λειτουργικότητα της αυτόματης επίλυσης με μια ευχάριστη και σαφή οπτική αναπαράσταση του εικονόσταυρου και της διαδικασίας λύσης.



Σχήμα 7.7: Σελίδα αυτόματης λύσης πριν την εκτέλεση.



Σχήμα 7.8: Σελίδα αυτόματης λύσης μετά την εκτέλεση.

7.3.5.4 Σελίδα Δημιουργίας Εικονόσταυρου

Η σελίδα δημιουργίας εικονόσταυρου σχεδιάστηκε με έμφαση στη λειτουργικότητα. Περιλάμβανε ένα διαδραστικό πλέγμα που επιτρέπει στους χρήστες να κάνουν αλλαγές, να ορίσουν προτιμήσεις και να περάσουν στην αυτόματη λύση μόλις ολοκληρώσουν τη δημιουργία.

7.4 Σύνοψη

Ο σχεδιασμός της εφαρμογής ξεκίνησε από μια ανεπίσημη, δημιουργική διαδικασία σε χαρτί και ολοκληρώθηκε με λεπτομερή ψηφιακή αποτύπωση στο *Figma*. Η συνδυαστική χρήση εξαρτημάτων, η ανάπτυξη πρωτοτύπων υψηλής πιστότητας και η εστίαση στη φιλικότητα προς τον χρήστη δημιούργησαν μια συνεπή και λειτουργική βάση για την περαιτέρω ανάπτυξη της εφαρμογής.

Κεφάλαιο 8ο: Στήσιμο Σελίδων, Μοντέλων και Κύριων Γραφικών

8.1 Εισαγωγή

Σε αυτό το τμήμα περιγράφεται πώς οργανώθηκε αρχικά το περιβάλλον του προγράμματος, συμπεριλαμβανομένης της ρύθμισης των βασικών πακέτων και της δημιουργίας των σελίδων με τη βοήθεια δρομολογητή. Μέσα από αυτή τη διαδικασία, μπήκαν οι θεμέλιοι λίθοι που καθορίζουν τη μελλοντική δομή και λειτουργικότητα της εφαρμογής.

Αφού εγκαταστάθηκαν τα αρχικά πακέτα στο περιβάλλον του *Android Studio*, ακολούθησε η δημιουργία των αρχείων που αντιστοιχούν στις σελίδες της εφαρμογής, με σημείο εκκίνησης τα προσχέδια που σχεδιάστηκαν προηγουμένως. Στη συνέχεια, με την προσθήκη ενός δρομολογητή σελίδων, εξασφαλίστηκε η δυνατότητα εύκολης μετάβασης του χρήστη ανάμεσα στα διάφορα τμήματα της εφαρμογής, επιτρέποντας τη δομή να αναπτυχθεί σταδιακά και με ορθό τρόπο.

8.2 Πλοήγηση

8.2.1 Εισαγωγή

Αρχικά, έπρεπε να γίνει το πρώτο στήσιμο του προγράμματος. Με τη δημιουργία του προγράμματος στο *Android Studio*, η διαδικασία ξεκίνησε με την εισαγωγή βασικών και απαραίτητων πακέτων που ήταν γνωστό ότι θα χρειάζονταν στη συνέχεια.

Με βάση τα σχέδια των σελίδων που είχαν δημιουργηθεί, όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, έγινε η δημιουργία των αρχείων των σελίδων και το αρχικό στήσιμο αυτών με έναν δρομολογητή. Έτσι, δημιουργήθηκε η βάση πάνω στην οποία θα αναπτυχθούν στη συνέχεια τα πιο σημαντικά κομμάτια της εφαρμογής.

8.2.2 Δρομολογητής Σελίδων

Ο δρομολογητής σελίδων διαδραματίζει ζωτικό ρόλο στην ομαλή λειτουργία και πλοήγηση της εφαρμογής. Ένας σωστά διαμορφωμένος δρομολογητής επιτρέπει την εύκολη μετάβαση μεταξύ των σελίδων και διασφαλίζει ότι ο χρήστης έχει μια συνεπή εμπειρία κατά τη χρήση της εφαρμογής. Επιπλέον, ένας καλά σχεδιασμένος δρομολογητής μπορεί να βελτιώσει την απόδοση της εφαρμογής, μειώνοντας το φορτίο και την καθυστέρηση κατά την πλοήγηση.

Μετά τη δημιουργία του δρομολογητή, ακολούθησε ο λεπτομερής σχεδιασμός των επιμέρους σελίδων. Αυτό περιλάμβανε:

- Τη δημιουργία δομικών στοιχείων κάθε σελίδας, όπως κουμπιά, πεδία εισαγωγής δεδομένων και περιοχές προβολής πληροφοριών.
- Την προσεκτική τοποθέτηση των στοιχείων για να εξασφαλιστεί η λειτουργικότητα και η ευχρηστία της εφαρμογής.

- Την εφαρμογή προτύπων σχεδιασμού που ενισχύουν την εμπειρία του χρήστη, όπως συνεπή χρώματα, γραμματοσειρές και ευθυγράμμιση στοιχείων.

8.3 Σελίδες Εφαρμογής

8.3.1 Εισαγωγή

Η οργάνωση και η διαχείριση των σελίδων της εφαρμογής προϋποθέτουν ένα πρώτο στάδιο εγκατάστασης και ρύθμισης βασικών δομών και αρχείων. Σε αυτό το εισαγωγικό βήμα, πραγματοποιήθηκε η αρχική διαμόρφωση του προγράμματος και η εγκατάσταση των απαραίτητων πακέτων, προετοιμάζοντας το έδαφος για την ανάπτυξη των κυρίων λειτουργιών.

Παράλληλα, αναπτύχθηκε η βασική αρχιτεκτονική των σελίδων, αξιοποιώντας τον σχεδιασμό που είχε ήδη υλοποιηθεί στο προηγούμενο στάδιο. Με αυτόν τον τρόπο, οι επιμέρους σελίδες της εφαρμογής απέκτησαν τη δική τους προγραμματιστική οντότητα, ενώ ο δρομολογητής ανέλαβε να ενορχηστρώσει τη σύνδεσή τους και την ομαλή εναλλαγή του χρήστη μεταξύ τους. Αυτή η πρακτική έβαλε τα θεμέλια για την επερχόμενη ενσωμάτωση σημαντικών χαρακτηριστικών, όπως η διαχείριση δεδομένων και η υλοποίηση του κεντρικού αλγορίθμου επίλυσης.

8.3.2 Σελίδα Λίστας Εικονόσταυρων

8.3.2.1 Εισαγωγή

Η σελίδα αυτή είναι το αρχικό σημείο εισόδου του χρήστη και παρέχει έναν οργανωμένο κατάλογο με όλα τα διαθέσιμα παζλ. Ο σχεδιασμός της είναι βασισμένος στη λειτουργικότητα και τη διαδραστικότητα.

8.3.2.2 Προσαρμοσμένος Σχεδιασμός για Κινητά και Web

Η διάταξη της σελίδας προσαρμόζεται δυναμικά στη συσκευή:

- Στις κινητές συσκευές, χρησιμοποιείται κάθετη διάταξη με εύκολη κύλιση.
- Στη web έκδοση, τα δεδομένα οργανώνονται σε πλέγμα για καλύτερη αξιοποίηση του διαθέσιμου χώρου.

8.3.2.3 Δημιουργία Νέου Παζλ

Ένα κουμπί δράσης (*floating action button*) επιτρέπει στον χρήστη να δημιουργήσει άμεσα ένα νέο παζλ. Αυτή η λειτουργία έχει σχεδιαστεί για να είναι εύκολα προσβάσιμη και να διευκολύνει την αλληλεπίδραση του χρήστη.

8.3.2.4 Προεπισκόπηση Παζλ

Κάθε στοιχείο της λίστας περιλαμβάνει προεπισκόπηση του παζλ, παρέχοντας οπτική αναπαράσταση, τίτλο και, αν υπάρχουν, σχετικές σημειώσεις. Αυτή η λειτουργία διευκολύνει την επιλογή του παζλ.

8.3.3 Σελίδα Επίλυσης Εικονόσταυρου

8.3.3.1 Εισαγωγή

Η σελίδα επίλυσης παρέχει ένα περιβάλλον εστιασμένο στη λύση του παζλ, προσφέροντας πλούσιες δυνατότητες για ανάλυση και αλληλεπίδραση.

8.3.3.2 Προσαρμογή για Κινητά και Web

Η διάταξη της σελίδας επίλυσης έχει σχεδιαστεί ώστε να προσαρμόζεται δυναμικά στη συσκευή που χρησιμοποιείται, εξασφαλίζοντας βέλτιστη εμπειρία χρήσης τόσο σε κινητές συσκευές όσο και σε περιβάλλον web.

8.3.3.2.1 Διάταξη για κινητές συσκευές

Στις κινητές συσκευές, η εφαρμογή λαμβάνει υπόψη το μέγεθος της οθόνης και τοποθετεί τόσο το πάζλ όσο και το κουμπί *«SOLVE»* εντός του ορατού χώρου. Με αυτόν τον τρόπο, ο χρήστης μπορεί να αλληλεπιδράσει με όλα τα απαραίτητα στοιχεία χωρίς να απαιτείται κύλιση. Αυτό επιτυγχάνεται με προσεκτικό υπολογισμό του διαθέσιμου ύψους, συμπεριλαμβανομένων των *padding* της οθόνης, της μπάρας εργαλείων και άλλων γραφικών στοιχείων.

8.3.3.2.2 Διάταξη για web έκδοση

Στη web έκδοση, η διάταξη διαφοροποιείται, καθώς το κουμπί *«SOLVE»* τοποθετείται δίπλα στο πάζλ, αξιοποιώντας την οριζόντια διάσταση της οθόνης. Σε αυτή την περίπτωση:

- Το πάζλ παραμένει ορατό χωρίς την ανάγκη κύλισης.
- Το κουμπί *«SOLVE»* είναι εύκολα προσβάσιμο λόγω της εγγύτητάς του με το πλέγμα.

Αυτή η προσέγγιση βελτιώνει την ευχρηστία και επιτρέπει στους χρήστες να εστιάσουν στην επίλυση του πάζλ, χωρίς να αποσπώνται από ανάγκες πλοήγησης.

8.3.4 Βήματα Επίλυσης

Ένας συρόμενος ελεγκτής (*slider*) επιτρέπει τη μετακίνηση μπρος και πίσω στα βήματα της λύσης. Αυτή η λειτουργία ενισχύει την κατανόηση της διαδικασίας λύσης, παρέχοντας πλήρη έλεγχο στον χρήστη.

8.3.5 Στατιστικά και Ρυθμίσεις

Η ενότητα των στατιστικών και των ρυθμίσεων στη σελίδα επίλυσης εικονόσταυρου παρέχει στον χρήστη λεπτομερή έλεγχο της διαδικασίας επίλυσης και χρήσιμες πληροφορίες για την παρακολούθηση της προόδου και της αποδοτικότητας.

8.3.5.1 Στατιστικά Στοιχεία

Τα στατιστικά προσφέρουν μια ολοκληρωμένη επισκόπηση της απόδοσης κατά την επίλυση:

- **Διάρκεια παζλ:** Συνολικός χρόνος που έχει περάσει από την έναρξη της επίλυσης.
- **Συνολικά βήματα:** Αριθμός βημάτων που πραγματοποιήθηκαν για την επίλυση του πάζλ.
- **Γραμμές που ελέγχθηκαν:** Συνολικός αριθμός γραμμών που εξετάστηκαν από τον αλγόριθμο.
- **Κουτάκια που ελέγχθηκαν:** Ο συνολικός αριθμός κουτιών που εξετάστηκαν κατά την επίλυση, περιλαμβάνοντας:
  - **Συνολικά:** Όλα τα κουτάκια που πέρασαν από έλεγχο.
  - **Δευτερευόντως:** Κουτάκια που εξετάστηκαν πέραν του κύριου αλγορίθμου λόγω επαναληπτικών διαδικασιών.
- **Δεδομένα προσωρινής μνήμης:** Πληροφορίες για τον αριθμό δεδομένων που αποθηκεύτηκαν προσωρινά κατά την επίλυση.

8.3.5.2 Ρυθμίσεις

Οι ρυθμίσεις παρέχουν στον χρήστη τη δυνατότητα προσαρμογής της εφαρμογής:

- **Αποθήκευση δεδομένων προσωρινής μνήμης:** Ενεργοποίηση ή απενεργοποίηση της προσωρινής μνήμης για αποθήκευση δεδομένων.
- **Ταξινόμηση γραμμών μέσω ενδείξεων:** Ενεργοποιεί την ταξινόμηση γραμμών με βάση το άθροισμα των ενδείξεων.
- **Μέτρηση κουτιών που ελέγχθηκαν:** Καταγραφή του αριθμού των κουτιών που ελέγχθηκαν.
- **Απομόνωση παράλληλων διεργασιών:** Ρύθμιση του αριθμού παράλληλων διεργασιών μέσω αριθμητικού ελεγκτή.
- **Επισήμανση νέων τιμών:** Εμφάνιση πρόσφατα συμπληρωμένων κουτιών με έντονο χρώμα.
- **Επεξεργασία μέσω *drag*:**
  - **Ενημέρωση στο τέλος της μετακίνησης (*onPanEnd*):** Τα κουτιά ενημερώνονται μετά τη μετακίνηση.
  - **Ενημέρωση κατά την κίνηση (*onPan*):** Τα κουτιά ενημερώνονται σε πραγματικό χρόνο.

Οι ρυθμίσεις παρουσιάζονται με τη μορφή διακοπτών (*switches*) ή αριθμητικών ελεγκτών, ανάλογα με τη φύση τους, διασφαλίζοντας ευκολία στη χρήση και άμεση προσαρμογή.

8.3.6 Σελίδα Δημιουργίας Εικονόσταυρου

8.3.6.1 Εισαγωγή

Η σελίδα δημιουργίας επιτρέπει την εξατομίκευση και τη δημιουργία νέων παζλ με εργαλεία που προσφέρουν ευκολία και ακρίβεια.

8.3.6.2 Ευελιξία Σχεδιασμού για Κινητά και Web

Η διάταξη προσαρμόζεται σε κάθε πλατφόρμα, προσφέροντας συρόμενους ελεγκτές και πλέγμα που ανταποκρίνεται σε άγγιγμα ή ποντίκι.

8.3.6.3 Προεπισκόπηση και Αυτόματη Ενημέρωση Ενδείξεων

Οι ενδείξεις ενημερώνονται δυναμικά με βάση τις αλλαγές στο πλέγμα, διασφαλίζοντας ότι το τελικό αποτέλεσμα αντανακλά τις επιλογές του χρήστη.

8.3.6.4 Εργαλεία Σχεδίασης και Διαγραφής

Τα κουτιά του πλέγματος μπορούν να συμπληρωθούν ή να διαγραφούν με απλές κινήσεις, ενώ οι αλλαγές αντικατοπτρίζονται άμεσα στις ενδείξεις του πάζλ.

8.3.6.5 Συνοψίζοντας

Οι τρεις κύριες σελίδες της εφαρμογής προσφέρουν ολοκληρωμένες δυνατότητες που καλύπτουν τη δημιουργία, τη λύση και την προβολή πάζλ. Η προσαρμογή για κινητές συσκευές και web εξασφαλίζει ενιαία εμπειρία χρήστη, ανεξαρτήτως πλατφόρμας.

8.4 Βάση Δεδομένων

8.4.1 Εισαγωγή

Η βάση δεδομένων αποτελεί το θεμέλιο της εφαρμογής για τη διαχείριση και την αποθήκευση δεδομένων που σχετίζονται με τα παζλ εικονόσταυρων. Το σύστημα έχει σχεδιαστεί ώστε να είναι ελαφρύ, γρήγορο και ενσωματωμένο στην εφαρμογή, επιτρέποντας άμεση πρόσβαση στα δεδομένα χωρίς εξάρτηση από εξωτερικές υπηρεσίες ή σύνδεση στο διαδίκτυο. Ο σχεδιασμός της δομής της βάσης συνδυάζει αποδοτικότητα και επεκτασιμότητα, υποστηρίζοντας μια σειρά λειτουργιών και χαρακτηριστικών που εξυπηρετούν την εμπειρία χρήστη.

8.4.2 Επιλογή Τύπου Βάσης

Η απόφαση να αποθηκεύονται όλα τα δεδομένα τοπικά εντός της εφαρμογής λήφθηκε με βάση δύο βασικούς παράγοντες:

- Τη βελτίωση της ταχύτητας φόρτωσης.
- Την απλότητα στη διαχείριση δεδομένων.

Καθώς η εργασία δεν απαιτεί την υποστήριξη εξατομικευμένων λογαριασμών ή προφίλ χρηστών, η χρήση εξωτερικών διακομιστών κρίθηκε περιττή. Αυτή η επιλογή εξασφαλίζει την άμεση διαθεσιμότητα των παζλ και τη διατήρηση μιας απλής, ανεξάρτητης αρχιτεκτονικής.

8.4.3 Επιλογή Έτοιμων Εικονόσταυρων

Τα εικονόσταυρα που χρησιμοποιήθηκαν προέρχονται από διαφορετικά επίπεδα δυσκολίας, ώστε να εξεταστεί η απόδοση του αλγορίθμου σε προβλήματα ποικίλης πολυπλοκότητας. Όλα προέρχονται από την ιστοσελίδα του Γιάν Γουόλτερ, είναι πάζλ που τους έχουν δωθεί άδεια για αναδιαμονή [9], και λύνονται με τη μέθοδο επίλυσης ανά γραμμή. Συγκεκριμένα, τα παζλ αυτά είναι:

1. **Dancer** (5x10)
2. **Cat** (8x8)
3. **Probably Not** (34x34)
4. **Swing** (45x45)

8.4.4 Δημιουργία Μοντέλων Εικονόσταυρων

8.4.4.1 Το Μοντέλο Nonogram

Το κύριο μοντέλο της εφαρμογής είναι το *Nonogram*, το οποίο αντιπροσωπεύει ένα παζλ. Περιέχει τις βασικές πληροφορίες, όπως:

- Μοναδικό αναγνωριστικό (`id`).
- Ενδείξεις (`clues`).
- Προαιρετικά στοιχεία, όπως σημειώσεις (`note`) και πρόσθετες πληροφορίες (`info`).

Το *Nonogram* παρέχει βασικές λειτουργίες, όπως:

- Πρόσβαση στο πλάτος και το ύψος του παζλ μέσω των μεθόδων `width` και `height`, οι οποίες βασίζονται στις ενδείξεις.
- Έλεγχος κατάστασης δημοσίευσης (`isPublished`) ή μοναδικότητας (`isUnique`).
- Δημιουργία μιας κενής λύσης (`emptySolution`) με το σύμβολο `?` για μη συμπληρωμένα κουτάκια.

Αυτή η ευέλικτη δομή επιτρέπει τη δυναμική διαχείριση διαφορετικών τύπων παζλ και τη γρήγορη ενσωμάτωσή τους στην εφαρμογή.

8.4.4.2 Το Μοντέλο Clues

Το μοντέλο *Clues* περιγράφει τις ενδείξεις του παζλ, διαχωρίζοντάς τις σε:

- Ενδείξεις γραμμών (`rows`).
- Ενδείξεις στηλών (`columns`).

Αυτό επιτρέπει την εύκολη πρόσβαση στα δεδομένα και τον υπολογισμό του αριθμού γραμμών και στηλών μέσω των μεθόδων `rowLength` και `columnLength`. Το μοντέλο υποστηρίζει επίσης τη μετατροπή σε *JSON*, καθιστώντας εύκολη την αποθήκευση και ανάκτηση δεδομένων.

8.4.4.3 Το Μοντέλο NonogramInfo

Το *NonogramInfo* παρέχει προαιρετικές πληροφορίες για το παζλ, όπως:

- Τίτλο (`title`).
- Συγγραφέα (`author`).
- Πνευματικά δικαιώματα (`copyright`).
- Περιγραφή (`description`).

Αυτό το μοντέλο ενισχύει την εμπειρία του χρήστη, προσφέροντας επιπλέον λεπτομέρειες για κάθε παζλ.

8.4.4.4 Το Μοντέλο SolutionStep

Το μοντέλο *SolutionStep* περιγράφει κάθε βήμα στη διαδικασία επίλυσης ενός παζλ. Περιλαμβάνει:

- Την τρέχουσα κατάσταση της λύσης (`currentSolution`).
- Εξήγηση του βήματος (`explanation`).
- Θέσεις των νέων συμπληρωμένων κουτιών (`newFilledBoxes`).
- Προαιρετικά στοιχεία, όπως τον άξονα (`axis`) και τη θέση της γραμμής (`lineIndex`).

8.4.4.5 Το Μοντέλο SolverSettings

Το *SolverSettings* παρέχει τις ρυθμίσεις για τη λειτουργία του αλγορίθμου επίλυσης, περιλαμβάνοντας παραμέτρους όπως:

- Διατήρηση προσωρινών δεδομένων (`keepCacheData`).
- Καταμέτρηση ελεγχόμενων κουτιών (`countCheckedBoxes`).

- Διαχείριση παράλληλων διεργασιών (`isolateConcurrent`).
- Επισήμανση νέων τιμών (`highlightNewFilledBoxes`).

Αυτό το μοντέλο προσφέρει υψηλή προσαρμοστικότητα, επιτρέποντας στους χρήστες να τροποποιούν τη συμπεριφορά του λύτη ανάλογα με τις ανάγκες τους.

8.5 Κωδικοποίηση Κύριων Γραφικών

8.5.1 Εισαγωγή

Η δημιουργία των κύριων γραφικών της εφαρμογής αποτέλεσε έναν από τους πιο κρίσιμους τομείς της ανάπτυξης, καθώς η σωστή απεικόνιση του πλέγματος και των στοιχείων του παζλ είναι απαραίτητη για την ομαλή λειτουργία και τη βέλτιστη εμπειρία χρήστη. Η διαδικασία αυτή απαιτούσε μια προσεγμένη προσέγγιση, καθώς έπρεπε να καλύψει τόσο τις αισθητικές ανάγκες της εφαρμογής όσο και τις υπολογιστικές απαιτήσεις, εξασφαλίζοντας τη συμβατότητα με διάφορες συσκευές και μεγέθη οθόνης.

Στο κεφάλαιο αυτό περιγράφεται η διαδικασία δημιουργίας και βελτίωσης του πλέγματος και των στοιχείων του παζλ, καθώς και η τελική ένωση αυτών των δύο συστατικών για την απόδοση ενός συνεκτικού και λειτουργικού γραφικού περιβάλλοντος. Οι σχεδιαστικές και τεχνικές προκλήσεις που προέκυψαν αντιμετωπίστηκαν μέσα από διαδοχικές δοκιμές και την εφαρμογή βέλτιστων πρακτικών.

Η υλοποίηση οργανώθηκε σε διακριτά στάδια:

- **Υπολογισμός Διαστάσεων Πλέγματος:** Ο καθορισμός των διαστάσεων των κουτιών του πλέγματος βάσει των δεδομένων του παζλ και των διαστάσεων της οθόνης.
- **Δημιουργία του Πλέγματος:** Η αρχική υλοποίηση με GridView και η τελική βελτιωμένη προσέγγιση με Custom Painter.
- **Σχεδιασμός Στοιχείων Παζλ:** Η απεικόνιση των στοιχείων (*clues*) στις άκρες του πλέγματος.
- **Ένωση Γραφικών Πλέγματος και Στοιχείων:** Η συγχώνευση των παραπάνω για την ολοκληρωμένη εμφάνιση του παζλ.

Με την ολοκλήρωση της διαδικασίας, το τελικό αποτέλεσμα ήταν ένα ευέλικτο, αποδοτικό και αισθητικά ελκυστικό περιβάλλον για τη διαχείριση και την επίλυση παζλ.

8.5.2 Γραφικά Πλέγματος Παζλ

8.5.2.1 Εισαγωγή

Το πλέγμα του εικονόσταυρου είναι ένα σύνολο από κουτιά, στοιβαγμένα σαν έναν δισδιάστατο πίνακα. Κάθε κουτί έχει το δικό του *state*, το οποίο είτε θα προέρχεται από ένα έτοιμο παζλ είτε θα δίνεται από τον χρήστη κατά τη δημιουργία του παζλ.

Καθώς το ίδιο το Flutter δεν περιέχει κάποιο ήδη πακέτο που να δίνει ακριβώς το γραφιστικό που ψάχνουμε, όλο έγινε από την αρχή με καθαρό κώδικα σε Flutter. Για την προγραμματιστική ανάπτυξή του, όλο το πλέγμα χωρίστηκε σε δύο κομμάτια:

- Το ίδιο το κουτάκι.
- Το πλέγμα που εμφανίζει πολλαπλά αντίγραφα αυτού του ενός κουτιού.

8.5.3 Σχεδιασμός Κουτιού

Το εργαλείο που χρησιμοποιήθηκε για τη δημιουργία του κουτιού και του πλέγματος ήταν το *Custom Painter* του Flutter. Με αυτό το εργαλείο, σχεδιάστηκαν τα κελιά και τα δεδομένα τους. Η λογική του *Custom Painter* μοιάζει με το *Turtle Graphics*, όπου καθοδηγούμε μια εικονική χελώνα να ξεκινήσει ή να σταματήσει το γράψιμο, να προχωρήσει ευθεία ή να στρίψει, και να επιλέξει το χρώμα και το φάρδος της γραμμής που γράφει.

Χρησιμοποιώντας το *Custom Paint*, δημιουργήθηκε ένα κουτάκι σχεδιάζοντας τέσσερις γραμμές μήκους όσο της παραμέτρου `Side`. Στήθηκε ένας Painter στον οποίο δόθηκαν δεδομένα για το χρώμα (μαύρο), το πλάτος της γραμμής (1 πίξελ) και το στυλ του (`PaintingStyle.fill` για να ζωγραφίσει μόνο το περίγραμμα). Στη συνέχεια, δόθηκαν οι εντολές για τη διαδρομή που θα ακολουθήσει ο Painter για να καταγράψει τα γραφικά. Συγκεκριμένα:

- Ξεκινά από το σημείο $(0, 0)$.
- Κινείται δεξιά στο σημείο $(\text{side}, 0)$.
- Μεταβαίνει κάτω στο $(\text{side}, \text{side})$.
- Επιστρέφει στο $(0, \text{side})$.
- Τέλος, επιστρέφει στο αρχικό σημείο $(0, 0)$.

8.5.3.1 Διαφορετικά Στάδια Κουτιού

Έπειτα, προστέθηκαν και τα υπόλοιπα στάδια που μπορούν να έχουν αυτά τα κουτάκια: βαμμένα, άδεια/διαγραμμένα (με ένα ”X” μέσα) και κενά (λευκά) των οποίων δεν γνωρίζουμε το στάδιο.

Για το διαγραμμένο κουτάκι, χρησιμοποιήθηκε το ίδιο στυλ και διαδρομή με το αρχικό, άδειο κουτάκι, με την προσθήκη επιπλέον γραμμών που δημιουργούν ένα ”X”.

8.5.4 Σχεδιασμός Πλέγματος Κουτιών

8.5.4.1 Δεδομένα που θα χρειαστούν για τη σχεδίαση

Ο σχεδιασμός του πλέγματος βασίστηκε στον τρόπο με τον οποίο τα δεδομένα μεταφέρονται στη σελίδα. Για τη δημιουργία του πλέγματος απαιτούνται τα εξής δεδομένα:

- Ο αριθμός των κουτιών στον οριζόντιο και κάθετο άξονα.
- Η λύση που πρέπει να εμφανιστεί (ως *Solution String*).

Τα δεδομένα αυτά καθορίζονται από τα μοντέλα των παζλ, όπως περιγράφονται στο Κεφάλαιο 8.4.3. Ο αριθμός των οριζόντιων και κάθετων κουτιών υπολογίζεται από το μέγεθος των λιστών στοιχείων (*clues*). Το πλάτος του παζλ προκύπτει από τον αριθμό των στηλών, ενώ το ύψος από τον αριθμό των γραμμών.

Η λύση κάθε παζλ αποθηκεύεται σε ένα *Solution String*, το οποίο αποτελεί μια συνεχόμενη ακολουθία χαρακτήρων. Η κατανομή των δεδομένων γίνεται από αριστερά προς δεξιά ανά γραμμή. Για παράδειγμα, σε ένα παζλ 5x5, το *Solution String* περιέχει 25 χαρακτήρες, όπου οι πρώτοι 5 αντιστοιχούν στην πρώτη γραμμή, οι επόμενοι 5 στη δεύτερη, και ούτω καθεξής. Αυτή η προσέγγιση εξασφαλίζει ότι τα δεδομένα των κουτιών διαβάζονται και αποδίδονται με ακρίβεια.

8.5.4.2 Υπολογισμός Διαστάσεων Πλέγματος

Για την υλοποίηση του πλέγματος του εικονόσταυρου, ο υπολογισμός του πλάτους και του ύψους κάθε κουτιού βασίζεται στις παρακάτω παραμέτρους:

- Το πλάτος της οθόνης, διαιρεμένο με τον αριθμό των στηλών του παζλ.
- Το ύψος της οθόνης, διαιρεμένο με τον αριθμό των γραμμών του παζλ.

Η μικρότερη από τις δύο τιμές χρησιμοποιείται για να εξασφαλιστεί ότι το πλέγμα προσαρμόζεται πλήρως στις διαστάσεις της οθόνης.

Επιπλέον, λαμβάνεται υπόψη η διαθεσιμότητα χώρου για επιπλέον γραφικά, όπως ολισθητές (*sliders*) και κουμπιά αλληλεπίδρασης. Αυτό διασφαλίζει ότι ο χρήστης μπορεί να πλοηγηθεί άνετα στο περιβάλλον της εφαρμογής, ανεξάρτητα από το μέγεθος της συσκευής.

8.5.4.3 Αρχική Προσέγγιση με GridView

Η αρχική υλοποίηση του πλέγματος έγινε με τη χρήση του *GridView*, ενός *widget* της Flutter που διευκολύνει την προβολή στοιχείων σε μορφή πλέγματος. Το *GridView* δημιουργεί αυτόματα το απαιτούμενο πλέγμα, λαμβάνοντας παραμέτρους όπως:

- Ο αριθμός των στηλών.
- Το συνολικό πλήθος των στοιχείων.

Παρόλο που η χρήση του *GridView* διευκόλυνε την αρχική υλοποίηση, παρουσίασε σημαντικούς περιορισμούς. Η αυξημένη υπολογιστική απαίτηση προκάλεσε καθυστερήσεις, ιδιαίτερα κατά την πλοήγηση στη σελίδα και την απόδοση μεγάλων παζλ. Ως αποτέλεσμα, το *GridView* κρίθηκε ακατάλληλο και αντικαταστάθηκε.

8.5.4.4 Βελτιωμένη Προσέγγιση με Custom Painter

Η τελική υλοποίηση του πλέγματος έγινε με τη χρήση του *Custom Painter*, ενός εργαλείου που παρέχει μεγαλύτερη ευελιξία και απόδοση.

Για την απόδοση του πλέγματος, κάθε κουτί σχεδιάστηκε ως ένα ανεξάρτητο αντικείμενο:

- **Εξωτερικός βρόχος:** Δημιουργία γραμμών του πλέγματος.
- **Εσωτερικός βρόχος:** Δημιουργία κουτιών για κάθε γραμμή.

Κάθε κουτί σχεδιάζεται ξεκινώντας από μία αρχική συντεταγμένη (x, y), η οποία προσαρμόζεται δυναμικά βάσει της θέσης του κουτιού στο πλέγμα. Το πρώτο κουτί τοποθετείται στην πάνω αριστερή γωνία του πλέγματος, ενώ κάθε επόμενο τοποθετείται δεξιά ή κάτω, ανάλογα με τη σειρά του.

Για παράδειγμα:

- Το πρώτο κουτί της δεύτερης γραμμής τοποθετείται στο $(\text{side}, 0)$.
- Το δεύτερο κουτί της δεύτερης γραμμής τοποθετείται στο $(\text{side}, \text{side})$.

8.5.4.5 Συμπέρασμα

Η μετάβαση από το *GridView* στο *Custom Painter* προσέφερε πολλαπλά οφέλη:

- Βελτιωμένη απόδοση για μεγάλου μεγέθους παζλ.
- Δυνατότητα προσαρμοσμένης απόδοσης και δυναμικής διάταξης.
- Μείωση καθυστερήσεων κατά την πλοήγηση και τη φόρτωση δεδομένων.

Η χρήση του *Custom Painter* απέδειξε τη σημασία της επιλογής εργαλείων που προσαρμόζονται στις ανάγκες της εφαρμογής, επιτυγχάνοντας υψηλή απόδοση και ευελιξία.

8.5.5 Γραφικά Στοιχείων Παζλ

8.5.5.1 Σχεδιασμός Στοιχείων

Ο σχεδιασμός των στοιχείων είχε σημαντική πολυπλοκότητα, καθώς έπρεπε να συνδυάσει τον υπολογισμό για την ομοιόμορφη και σωστή τοποθέτησή τους, λαμβάνοντας υπόψη το μέγεθος του πλέγματος.

Τα στοιχεία βρίσκονται σε δύο θέσεις του πλέγματος:

- Στην αριστερή πλευρά (οριζόντια λίστα).
- Στην πάνω πλευρά (κάθετη λίστα).

Και στις δύο θέσεις, το γραφικό είναι το ίδιο, αποτελούμενο από μια λίστα αριθμών. Η στοίχιση των στοιχείων γίνεται ως εξής:

- Αριστερή λίστα: Από αριστερά προς τα δεξιά.
- Πάνω λίστα: Από πάνω προς τα κάτω.

Για το στήσιμο αυτών χρησιμοποιήθηκε ο ίδιος κώδικας, με τροποποιημένες παραμέτρους.

8.5.5.2 Υπολογισμός Μεγεθών

Το μέγεθος των κουτιών εξαρτάται από το μέγεθος του πλέγματος:

- Εντοπίζεται η λίστα με τα περισσότερα στοιχεία στις γραμμές και τις στήλες.
- Το μέγεθος αυτό προστίθεται στο πλάτος και το ύψος του πλέγματος.
- Καθορίζονται οι διαστάσεις για κάθε κουτί, ώστε να διατηρηθεί η αισθητική συνοχή.

8.5.6 Ένωση Γραφικών Πλέγματος και Στοιχείων Παζλ

Η ενοποίηση του πλέγματος του παζλ με τα γραφικά στοιχεία (*clues*) υλοποιήθηκε μέσω μίας απλής αλλά αποτελεσματικής παραμετροποίησης στα μεγέθη που ορίστηκαν ως περιορισμοί για τα γραφικά.

Τα στοιχεία και τα κουτιά του πλέγματος διατηρούν την ίδια διάσταση, γεγονός που διευκολύνει την ομοιόμορφη σχεδίαση και ευθυγράμμιση.

Για τον τελικό υπολογισμό του πλάτους του πλέγματος:

- Προστίθεται ο αριθμός των κουτιών στη γραμμή με τα περισσότερα στοιχεία (κάθετα).

Για το ύψος:

- Προστίθεται ο αριθμός των κουτιών στη στήλη με τα περισσότερα στοιχεία (οριζόντια).

Αυτή η προσέγγιση εξασφαλίζει ισορροπία και αισθητική συνοχή, προσφέροντας ταυτόχρονα τη λειτουργικότητα που απαιτείται για την παρουσίαση των δεδομένων του παζλ.

8.5.7 Διαχείριση Κατάστασης (State Management)

8.5.7.1 Εισαγωγή

Η διαχείριση της κατάστασης μιας εφαρμογής αποτελεί κρίσιμο παράγοντα για την απόδοσή της, ιδιαίτερα σε εφαρμογές που απαιτούν διαρκή ενημέρωση της προβολής με βάση τη δυναμική αλληλεπίδραση του χρήστη.

Στην παρούσα εργασία, η προσέγγιση για τη διαχείριση της κατάστασης εξελίχθηκε, ξεκινώντας από τα *Hooks* και καταλήγοντας στο πρότυπο *BLoC* (*Business Logic Component*).

8.5.7.2 Χρήση Hooks και BLoC

Αρχικά, η διαχείριση της κατάστασης του έργου βασίστηκε στα *Hooks*. Αυτή η επιλογή έγινε με στόχο την εύκολη και γρήγορη ενημέρωση της κατάστασης του παζλ κατά τη διάρκεια της λύσης, διατηρώντας την προβολή του χρήστη πάντα ενημερωμένη. Τα *Hooks* επελέγησαν λόγω:

- Της απλότητάς τους σε επίπεδο κώδικα.
- Της δυνατότητάς τους να συγκεντρώνουν τόσο την κατάσταση όσο και τις μεθόδους επηρεασμού της σε ένα ενιαίο σημείο.

Ωστόσο, κατά την υλοποίηση προέκυψαν σημαντικές δυσκολίες:

8.5.7.2.1 Περιορισμοί του Flutter Web

Το *Flutter* λειτουργεί ως εφαρμογή μονού νήματος (*single-threaded*), κάτι που σημαίνει ότι η ανανέωση της προβολής σε πραγματικό χρόνο απαιτεί τη χρήση *Isolates*. Αν και οι *Isolates* αποτελούν ισχυρό εργαλείο, το *Flutter Web* υποστηρίζει μόνο *Workers*, μια προηγούμενη και πιο περιορισμένη έκδοση των *Isolates*. Αυτοί οι περιορισμοί καθιστούσαν τη συνδυαστική χρήση εργαλείων διαχείρισης κατάστασης και *Workers* μη εφικτή.

8.5.7.2.2 Πολυπλοκότητα στη Διαχείριση του Γραφιστικού

Η λογική που απαιτείται για την ανανέωση της προβολής με βάση δεδομένα από τον *Worker* αποδείχθηκε πιο σύνθετη από το αναμενόμενο. Η συγκέντρωση της κατάστασης και των μεθόδων που την επηρεάζουν στον ίδιο κώδικα καθιστούσε το αποτέλεσμα δυσανάγνωστο και δύσκολο στη διαχείριση.

Λόγω αυτών των περιορισμών, αποφασίστηκε η μετάβαση από τα *Hooks* στο πρότυπο *BLoC*.

8.5.7.3 Βήματα Μετάβασης στο BLoC

Η μετάβαση στο πρότυπο *BLoC* πραγματοποιήθηκε μέσα από τα εξής βήματα:

8.5.7.3.1 Διαχωρισμός Λειτουργιών Solver

Όλες οι μέθοδοι που σχετίζονταν με τον αλγόριθμο λύσης (*Solver*) απομονώθηκαν σε ξεχωριστά αρχεία, διαχωρίζοντάς τες πλήρως από το σύστημα διαχείρισης κατάστασης.

8.5.7.3.2 Αναδιοργάνωση του State Management

Οι λειτουργίες που αφορούσαν την ανανέωση της προβολής μεταφέρθηκαν στο *Cubit* του *BLoC*. Παράλληλα:

- Το *state* της σελίδας τοποθετήθηκε σε ξεχωριστό αρχείο, το οποίο περιείχε αποκλειστικά τις πληροφορίες για το τι προβάλλεται στον χρήστη.
- Ο διαχωρισμός αυτός επέτρεψε την πλήρη ανεξαρτησία των λειτουργιών προβολής από τις λειτουργίες λογικής.

Με αυτή τη δομή, επιτεύχθηκε διαχωρισμός των λειτουργιών ως εξής:

- **State:** Καταγράφει μόνο την τρέχουσα κατάσταση της σελίδας, δηλαδή τι βλέπει ο χρήστης εκείνη τη στιγμή.
- **Cubit:** Διαχειρίζεται το πώς αλλάζει αυτή η κατάσταση, ενεργώντας ως γέφυρα μεταξύ των γεγονότων της εφαρμογής και της ενημέρωσης του *state*.

8.5.7.4 Πλεονεκτήματα της Μετάβασης

Η νέα προσέγγιση οδήγησε σε έναν πιο καθαρό και ευανάγνωστο κώδικα. Ειδικότερα:

- Το *BLoC* της σελίδας επιτυγχάνει αποτελεσματική και επεκτάσιμη διαχείριση κατάστασης.
- Διασφαλίζεται η αρμονική λειτουργία μεταξύ του *Solver*, του *Worker* και της διεπαφής χρήστη.
- Η πλήρης απομόνωση του *state* από τη λογική επιτρέπει την εύκολη τροποποίηση ή επαναχρησιμοποίηση των λειτουργιών σε μελλοντικά έργα.

Κεφάλαιο 9ο: Αυτόματος Λύτης

9.1 Εισαγωγή

Η ενότητα «Λύτης» παρουσιάζει αναλυτικά τη διαδικασία σχεδίασης και υλοποίησης του συστήματος που χρησιμοποιείται για την επίλυση εικονόσταυρων. Περιγράφεται αρχικά η ιδέα πίσω από την επιλογή του θέματος και ο τρόπος με τον οποίο αυτή υποστηρίζει την αποτελεσματικότητα και τη λειτουργικότητα της εφαρμογής.

Στη συνέχεια, επεξηγείται η δομή της βάσης δεδομένων και ο τρόπος κωδικοποίησης των παζλ, παρέχοντας πληροφορίες για την ταχύτητα φόρτωσης και την αποδοτικότητα του συστήματος. Εξετάζεται επίσης η σχεδίαση του γραφικού περιβάλλοντος χρήστη (UI), συμπεριλαμβανομένων των σελίδων επίλυσης, της λίστας παζλ, καθώς και της ειδικής σελίδας δημιουργίας παζλ.

Η ενότητα αναλύει διεξοδικά τον αλγόριθμο επίλυσης, με έμφαση στη μεθοδολογία ανάλυσης γραμμών και στηλών, τον έλεγχο συμβατότητας ενδείξεων και την παραμετροποίηση για βέλτιστη απόδοση. Τέλος, περιγράφεται η δυνατότητα δημιουργίας παζλ από τους χρήστες, διευκολύνοντας τη σχεδίαση, ρύθμιση και αποθήκευση αυτών των παζλ.

Όλος ο κώδικας της παρούσας εργασίας, καθώς και περαιτέρω πληροφορίες και μελλοντική εξέλιξη, είναι διαθέσιμα στο GitHub repository του προφίλ μου [10].

9.2 Χαρακτηριστικά

9.2.1 Εισαγωγή

Ο «Λύτης» εικονόσταυρων ενσωματώνει μια σειρά από καινοτόμες λειτουργίες που στοχεύουν στη βελτίωση της απόδοσης, της χρηστικότητας και της εκπαιδευτικής του αξίας. Τα χαρακτηριστικά του συστήματος περιγράφονται λεπτομερώς παρακάτω.

9.2.2 Αποδοτική Επίλυση

Η καρδιά του «Λύτη» βασίζεται σε αλγόριθμους υψηλής απόδοσης. Οι βασικές μέθοδοι περιλαμβάνουν:

- **Γραμμική μέθοδο ανάλυσης:** Αξιοποιούνται σειριακοί υπολογισμοί ανά γραμμή ή στήλη για την εύρεση λύσεων.
- **Παράλληλη επεξεργασία (*isolate concurrency*):** Υποστηρίζεται η εκτέλεση πολλαπλών διεργασιών σε διαφορετικά *isolates*, επιτρέποντας ταχύτερη επίλυση σε συστήματα πολλαπλών πυρήνων.

Οι τεχνικές λεπτομέρειες περιγράφονται σε επόμενες ενότητες.

9.2.3 Προσαρμοστικότητα Συστήματος

9.2.3.1 Διατήρηση Προσωρινών Δεδομένων

Ο «Λύτης» προσφέρει υψηλή προσαρμοστικότητα μέσω των παρακάτω ρυθμίσεων:

- Επιτρέπει επανεκκίνηση ή συνέχιση της διαδικασίας επίλυσης.
- Ρυθμίζει τον αριθμό παράλληλων διεργασιών μέσω του *isolateConcurrent*.

9.2.3.2 Επεξηγήσεις και Εκπαιδευτική Αξία

Ο «Λύτης» παρέχει επεξηγήσεις για κάθε βήμα της λύσης, π.χ.:

- Πώς υπολογίζονται οι δυνατές τοποθετήσεις.
- Ποια δεδομένα εξαιρούνται και γιατί.
- Ποιο είναι το τελικό αποτέλεσμα.

9.2.4 Διαδραστικότητα

Η διαδραστική διεπαφή περιλαμβάνει συρόμενο ελεγκτή (*slider*) που επιτρέπει στους χρήστες να μετακινηθούν μπρος-πίσω στα βήματα της λύσης.

9.2.5 Διαχείριση Πόρων

Με την ανάλυση γραμμών μέσω *clues (stack sorting)*, το σύστημα εξασφαλίζει ότι οι πόροι της συσκευής χρησιμοποιούνται αποδοτικά. Για παράδειγμα, οι γραμμές ή στήλες με μεγαλύτερη πληροφορία υπολογίζονται πρώτες.

9.2.6 Συγχρονισμός Δεδομένων και Εξοικονόμηση Χρόνου

Όλες οι πληροφορίες λύσης αποθηκεύονται και συγχρονίζονται, επιτρέποντας στους χρήστες να επιστρέψουν ή να συνεχίσουν τη λύση οποιαδήποτε στιγμή.

9.3 Κάλεσμα Μεθόδου Λύτη

9.3.1 Μέθοδος `solvePuzzle`

Η μέθοδος `solvePuzzle` ενεργοποιείται με το πάτημα του κουμπιού «Solve» στη σελίδα του παζλ. Αρχικά, η μέθοδος επιστρέφει ένα προκαθορισμένο αποτέλεσμα, π.χ.:

```
{
  "status": "Unsolved"
}
```

Αυτή η βασική υλοποίηση είχε ως κύριο στόχο την παρακολούθηση της προόδου του αλγορίθμου του λύτη κατά τη διάρκεια της ανάπτυξής του. Με αυτόν τον τρόπο, εξασφαλίστηκε ένα πρώτο επίπεδο δοκιμών και η δυνατότητα σταδιακής βελτίωσης της λειτουργικότητας. Οι βελτιώσεις περιγράφονται σε επόμενες ενότητες.

9.4 Βρόχος Επεξεργασίας Γραμμών

9.4.1 Εισαγωγή

Η κύρια στρατηγική του λύτη βασίζεται στην ανάλυση κάθε γραμμής και στήλης του παζλ, χρησιμοποιώντας έναν βρόχο που επεξεργάζεται τις γραμμές και τις στήλες με βάση ανανεωμένα δεδομένα. Για τον σκοπό αυτό, σχεδιάστηκε μια λίστα (σωρός), η οποία περιλαμβάνει τις γραμμές και τις στήλες προς επεξεργασία.

9.4.1.1 Βασικές Ανάγκες του Βρόχου

Ο βρόχος καλύπτει τις εξής ανάγκες:

- **Πληθυσμός λίστας:** Προσθήκη των γραμμών και στηλών στη λίστα.
- **Κριτήρια Τερματισμού:** Καθορισμός των συνθηκών ολοκλήρωσης της διαδικασίας.

9.4.1.2 Σχετικές Μέθοδοι

Οι ανάγκες αυτές υλοποιούνται μέσω των παρακάτω μεθόδων:

- `loopSides`: Υλοποιεί τον κύριο βρόχο επεξεργασίας.
- `initializeStackList`: Αναλαμβάνει την αρχικοποίηση της λίστας.
- `getNewStackElements`: Ανανεώνει τη λίστα με νέα δεδομένα.

9.4.2 Περιεχόμενο Λίστας (Σωρού)

9.4.2.1 Δομή και Πληροφορίες

Η λίστα περιλαμβάνει:

- Αριθμό θέσης γραμμής ή στήλης (π.χ., 0 έως μέγιστο).
- Τύπο άξονα (`row` ή `column`).

Η δομή της λίστας βασίζεται σε συλλογή από αντικείμενα `Map`, όπου:

- Το `key` αντιστοιχεί στον αριθμό γραμμής/στήλης.
- Το `value` υποδεικνύει τον άξονα (`NonoAxis`).

9.4.2.1.1 Πλεονεκτήματα Σχεδίασης

Η χρήση μεμονωμένων αντικειμένων `Map` αποφεύγει προβλήματα όπως η αντικατάσταση (override) δεδομένων μεταξύ γραμμών και στηλών.

9.4.2.1.2 Περιορισμοί

Η μέθοδος αυτή δεν είναι η πλέον βέλτιστη σε απόδοση, αλλά εξυπηρετεί αξιόπιστα την ορθή λειτουργία του αλγορίθμου.

9.4.3 Αρχικοποίηση Λίστας (Σωρού)

9.4.3.1 Αρχική Λύση

Η αρχικοποίηση γίνεται μέσω της μεθόδου `initializeStackList`.

Στην πρώτη, πιο απλή προσέγγιση, προστέθηκαν όλες οι γραμμές και όλες οι στήλες του παζλ με τη σειρά. Αυτό διασφάλιζε ότι ο αλγόριθμος θα επεξεργαστεί τουλάχιστον όλα τα δεδομένα που υπάρχουν αρχικά στο παζλ, εντοπίζοντας τα αρχικά στοιχεία τους.

Η διαδικασία αυτή επιτεύχθηκε μέσω δύο διαδοχικών βρόχων:

1. **Βρόχος Γραμμών:** Προσθήκη αντικειμένων `Map`, όπου το `key` είναι ο αριθμός γραμμής και το `value` είναι `NonoAxis.row`.
2. **Βρόχος Στηλών:** Παρόμοια διαδικασία για στήλες, με `value` `NonoAxis.column`.

9.4.3.2 Βελτιστοποίηση Αρχικοποίησης

Σε επόμενο στάδιο βελτιστοποίησης, η μέθοδος αναβαθμίστηκε με έναν ελαφρώς πιο έξυπνο αλγόριθμο. Αντί να προστίθενται οι γραμμές και οι στήλες απλώς με τη σειρά, έγινε παρατήρηση ότι γραμμές και στήλες με μεγαλύτερο άθροισμα στοιχείων έχουν υψηλότερη πιθανότητα να περιέχουν περισσότερα συμπληρωμένα δεδομένα. Για παράδειγμα:

- Σε ένα παζλ 50x50, μια γραμμή ή στήλη με στοιχεία 25 και 15 έχει περισσότερες πιθανότητες να είναι ”λύσιμη” από μια γραμμή ή στήλη με στοιχεία 2 και 3.

Με βάση αυτό το κριτήριο:

1. Υπολογίζονται τα αθροίσματα στοιχείων κάθε γραμμής και στήλης.
2. Τα δεδομένα ταξινομούνται (sort) φθίνουσα, από το μεγαλύτερο προς το μικρότερο άθροισμα.
3. Η λίστα (σωρός) δημιουργείται με προτεραιότητα από το μεγαλύτερο προς το μικρότερο.

Αν και αυτή η βελτίωση βελτιστοποιεί την απόδοση του αλγορίθμου, η διαφορά δεν είναι πάντα αισθητή. Για τον λόγο αυτό, ενσωματώθηκε ως προαιρετική λειτουργία. Ο χρήστης έχει τη δυνατότητα να την ενεργοποιεί ή να την απενεργοποιεί πριν την έναρξη της διαδικασίας επίλυσης, ανάλογα με τις ανάγκες του.

9.4.4 Ανανέωση Λίστας

9.4.4.1 Λόγος Ανανέωσης Λίστας

Κατά την εκτέλεση του αλγορίθμου, η συνεχής επανεξέταση των γραμμών και στηλών είναι απαραίτητη, καθώς προκύπτουν νέες πληροφορίες που αλλάζουν τη δυναμική του παζλ. Χωρίς αυτή τη διαδικασία, ο αλγόριθμος θα απαιτούσε όλες οι αλλαγές να επιλυθούν σε μία μόνο επανάληψη, κάτι που είναι εξαιρετικά απίθανο, ιδιαίτερα για μεγαλύτερα και πιο σύνθετα παζλ.

Η επανεξέταση γραμμών ή στηλών καθίσταται αναγκαία όταν αλλάζουν δεδομένα:

- Είτε συμπληρώνονται κελία.
- Είτε διαγράφονται εσφαλμένα στοιχεία.

Σε αυτές τις περιπτώσεις, η μέθοδος `getNewStackElements` εντοπίζει τις επηρεασμένες περιοχές και τις προσθέτει στη λίστα προς περαιτέρω ανάλυση.

9.4.5 Μέθοδος Ανάκτησης Νέων Γραμμών Προς Επεξεργασία (`getNewStackElements`)

9.4.5.1 Ρόλος και Σχεδιασμός της Μεθόδου

Η μέθοδος `getNewStackElements` είναι υπεύθυνη για τον εντοπισμό όλων των γραμμών και στηλών που επηρεάζονται από αλλαγές κατά την ανάλυση ενός στοιχείου. Αυτή η μέθοδος είναι θεμελιώδης για την αποδοτική λειτουργία του αλγορίθμου, καθώς επιτρέπει τη συνεχή εστίαση σε κρίσιμα σημεία του παζλ.

9.4.5.2 Λειτουργία της Μεθόδου

- Λαμβάνει ως παραμέτρους τον εξεταζόμενο άξονα (`row` ή `column`) και τους δείκτες (`indexes`) των επηρεασμένων στοιχείων.
- Δημιουργεί μια λίστα από επηρεασμένα στοιχεία χρησιμοποιώντας δομές `Map`, όπου:
  - Το `key` αντιπροσωπεύει τον δείκτη (`index`) του στοιχείου.
  - Το `value` ορίζει τον αντίθετο άξονα (π.χ., `column` για εξεταζόμενη γραμμή).

9.4.5.3 Παραδείγματα Υλοποίησης

- Αν εξετάζεται μια γραμμή, τα επηρεασμένα στοιχεία στις στήλες καταγράφονται με τρόπο που να εξασφαλίζει την ανανέωση.

- Τα δεδομένα που επιστρέφονται περιλαμβάνουν κάθε στήλη που διασταυρώνεται με τα επηρεασμένα στοιχεία της γραμμής.

Η μέθοδος `getNewStackElements` προσφέρει έναν απλό αλλά αποδοτικό τρόπο παρακολούθησης των αλλαγών, εξασφαλίζοντας ότι ο αλγόριθμος διατηρεί τη δυναμική του.

9.4.6 Ανανέωση Λίστας (Σωρού) με Νέα Στοιχεία

9.4.6.1 Τρόπος Ανανέωσης

Η ανανέωση της λίστας πραγματοποιείται αμέσως μετά την ολοκλήρωση της επεξεργασίας ενός στοιχείου από τον αλγόριθμο. Οι νέες πληροφορίες που προκύπτουν οδηγούν σε:

- Εντοπισμό των αλλαγών που επηρεάζουν άλλες γραμμές/στήλες.
- Προσθήκη των νέων στοιχείων στο τέλος της λίστας.

9.4.6.2 Αποφυγή Διπλοεγγραφών

Ένα σημαντικό χαρακτηριστικό είναι ότι η μέθοδος αποφεύγει τη διπλοεγγραφή στοιχείων στη λίστα. Αν κάποιο στοιχείο υπάρχει ήδη, δεν προστίθεται ξανά. Αυτό μειώνει τις περιττές επεξεργασίες, εξοικονομώντας χρόνο και πόρους.

9.4.7 Αποτυχημένες Προσπάθειες Βελτιστοποίησης

9.4.7.1 Αρχική Προσέγγιση: Μετακίνηση Στοιχείων

Σε πρώιμα στάδια, επιχειρήθηκε η μετακίνηση επηρεασμένων στοιχείων στην αρχή της λίστας. Η ιδέα ήταν να δοθεί προτεραιότητα σε κρίσιμες περιοχές. Ωστόσο:

- Αυξήθηκε η πολυπλοκότητα λόγω της αναδιάταξης.
- Η τυχαιότητα της συμβολής στη λύση δεν παρείχε σαφή οφέλη.

9.4.7.2 Εναλλακτική Στρατηγική: Τοποθέτηση σε Ενδιάμεση Θέση

Μια δεύτερη προσέγγιση προέβλεπε τη μεταφορά στοιχείων σε ενδιάμεσες θέσεις (π.χ., από τη 12η στην 6η). Παρά την προσπάθεια μείωσης του κόστους αναδιάταξης:

- Η αποδοτικότητα δεν βελτιώθηκε.
- Η επιπλέον πολυπλοκότητα οδήγησε σε αυξημένο χρόνο επεξεργασίας.

9.4.7.3 Συμπέρασμα

Η απλούστερη μέθοδος, κατά την οποία τα νέα στοιχεία προστίθενται στο τέλος της λίστας χωρίς αναδιάταξη, αποδείχθηκε η πιο αποδοτική. Η εμπειρία αυτή ανέδειξε τη σημασία της απλότητας στον σχεδιασμό αλγορίθμων και της συνεχούς δοκιμής πριν την υιοθέτηση σύνθετων στρατηγικών.

9.5 Μέθοδος Επεξεργασίας Γραμμών (loopSides)

9.5.1 Εισαγωγή

Η μέθοδος `loopSides` αποτελεί τον πυρήνα της λειτουργίας του αλγορίθμου για την επίλυση γραμμών και στηλών σε παζλ εικονόσταυρου. Σχεδιάστηκε για να επεξεργάζεται τα δεδομένα που σχετίζονται με κάθε γραμμή ή στήλη, να αναλύει την κατάσταση των κουτιών και να ενημερώνει την τρέχουσα λύση του παζλ, ανάλογα με τα αποτελέσματα της επεξεργασίας.

9.5.1.1 Εισαγωγή στη λειτουργία

Η μέθοδος δέχεται ως είσοδο πληροφορίες όπως οι ενδείξεις (clues) για τη γραμμή ή τη στήλη, το είδος του άξονα (γραμμή ή στήλη) και την τρέχουσα κατάσταση της λύσης. Η κύρια λειτουργία της είναι να εντοπίζει ποια κουτιά μπορούν να συμπληρωθούν ή να διαγραφούν με βάση τους κανόνες του παζλ.

9.5.1.2 Οργανωμένη δομή παρουσίασης

Στην ενότητα αυτή παρουσιάζονται:

- Η λειτουργία της μεθόδου.
- Οι βασικές διαδικασίες, όπως η ανάκτηση δεδομένων γραμμής, η ανάλυση πιθανών λύσεων, η ενημέρωση της λύσης και η διαχείριση του σωρού (λίστας επεξεργασίας).

9.5.2 Λειτουργία της Μεθόδου

Η `loopSides` εκτελεί τις παρακάτω διαδοχικές λειτουργίες για την επεξεργασία γραμμών ή στηλών του παζλ.

9.5.2.1 Αρχική ανάκτηση δεδομένων

Η πρώτη φάση περιλαμβάνει την ανάκτηση των δεδομένων της γραμμής ή της στήλης που πρόκειται να υποβληθεί σε επεξεργασία:

- Με τη χρήση της `getSolutionLine`, επιστρέφεται η τρέχουσα κατάσταση της γραμμής ή στήλης, αναπαριστώντας κάθε κουτί ως `1` (συμπληρωμένο), `0` (διαγραμμένο), ή `?` (κενό).
- Η μέθοδος `getCharIndexesOfQuestionMarks` εντοπίζει τις θέσεις των κουτιών που παραμένουν κενά.

9.5.2.2 Έλεγχος ολοκλήρωσης γραμμής

Η μέθοδος συγκρίνει τον αριθμό των συμπληρωμένων κουτιών με το άθροισμα των ενδείξεων (`clues`). Εάν η γραμμή είναι ολοκληρωμένη:

- Τα υπόλοιπα κενά κουτιά διαγράφονται.
- Η γραμμή ενημερώνεται και αποθηκεύεται.
- Η λίστα (σωρός) εμπλουτίζεται με τις επηρεασμένες γραμμές ή στήλες.

9.5.2.3 Διαδικασία για μη ολοκληρωμένες γραμμές

Εάν η γραμμή δεν είναι ολοκληρωμένη, τότε:

- Υπολογίζονται όλες οι πιθανές λύσεις με την `_getAllLinePossibleSolutions`.
- Οι λύσεις συγκρίνονται (ακραίες λύσεις από την αρχή και το τέλος) μέσω `_getSideMostSolution`.
- Τα κοινά στοιχεία μεταξύ των ακραίων λύσεων αναγνωρίζονται μέσω `_getSideMostSolutionsMatches`.

9.5.2.4 Ενημέρωση της λύσης

Μετά την ανάλυση:

- Τα κουτιά που μπορούν να συμπληρωθούν ή να διαγραφούν ενημερώνονται.
- Η λίστα (σωρός) ανανεώνεται με τις επηρεασμένες γραμμές και στήλες.
- Τα αποτελέσματα επιστρέφονται για περαιτέρω επεξεργασία.

9.5.3 Ανάκτηση Δεδομένων Γραμμής

9.5.3.1 Εισαγωγή

Η ανάκτηση δεδομένων γραμμής αποτελεί το πρώτο και θεμελιώδες βήμα στη λειτουργία της μεθόδου `loopSides`, διασφαλίζοντας ότι κάθε γραμμή ή στήλη του παζλ εικονόσταυρου θα υποβληθεί σε ανάλυση με ακρίβεια και σαφήνεια. Σε αυτό το στάδιο, πραγματοποιούνται δύο βασικές λειτουργίες:

1. Ανάκτηση της τρέχουσας κατάστασης της γραμμής ή της στήλης.
2. Εντοπισμός των μη συμπληρωμένων στοιχείων (?).

9.5.3.2 Ανάκτηση της Τρέχουσας Κατάστασης Γραμμής ή Στήλης

Η ανάκτηση της τρέχουσας κατάστασης γραμμής ή στήλης πραγματοποιείται μέσω της μεθόδου `getSolutionLine`, η οποία λαμβάνει ως εισόδους:

- Τη συνολική συμβολοσειρά της λύσης.
- Το πλάτος του παζλ (`nonogramWidth`).
- Τη θέση της γραμμής ή στήλης (`lineIndex`).
- Τον τύπο του άξονα (`NonoAxis.row` ή `NonoAxis.column`).

Ανάλογα με τον άξονα, εφαρμόζεται διαφορετική λογική, ώστε να ανακτηθεί σωστά η κατάσταση της γραμμής ή της στήλης.

9.5.3.2.1 Αλγόριθμος Ανάκτησης Γραμμής

Η μέθοδος `getSolutionLine` αναλύει τη συμβολοσειρά για να εξαγάγει την κατάσταση μιας γραμμής:

- **Χρήση της `split`:** Διαχωρίζει τη συμβολοσειρά της λύσης σε έναν πίνακα χαρακτήρων.
- **Επιλογή εύρους:** Υπολογίζεται το εύρος χαρακτήρων που αντιστοιχεί στη γραμμή:
  Αρχή = lineIndex $\times$nonogramWidth,
  Τέλος $=$ (lineIndex $+ 1$) $\times$ nonogramWidth.
- **Επανασύνθεση με `join`:** Τα στοιχεία ενώνονται σε μια συμβολοσειρά που αντιπροσωπεύει την κατάσταση της γραμμής.

9.5.3.2.1.1 Παράδειγμα:

Για `solution = "000011112222"` και `nonogramWidth = 4`, η γραμμή με `lineIndex = 0` επιστρέφει:

Αποτέλεσμα: ”0000”.

9.5.3.2.2 Αλγόριθμος Ανάκτησης Στήλης

Η ανάκτηση της κατάστασης μιας στήλης στο παζλ εικονόσταυρου διαφέρει από εκείνη της γραμμής, καθώς τα δεδομένα της στήλης είναι διασκορπισμένα σε συγκεκριμένα διαστήματα μέσα στη συνολική λύση (solution). Αντί να αποτελεί συνεχόμενη ακολουθία χαρακτήρων, η στήλη απαιτεί επεξεργασία μέσω επαναληπτικής διαδικασίας. Η μέθοδος `getSolutionLine` αναλαμβάνει να εξαγάγει τους χαρακτήρες της στήλης με ακρίβεια.

9.5.3.2.2.1 Βασική Δομή

Η μέθοδος ξεκινά με τη δημιουργία μιας κενής συμβολοσειράς (`columnSol`) για την αποθήκευση των χαρακτήρων της στήλης. Ένας επαναληπτικός βρόχος (`for loop`) χρησιμοποιείται για την εξαγωγή των χαρακτήρων από τη συνολική λύση.

9.5.3.2.2.1.1 Υπολογισμός Θέσεων των Χαρακτήρων

Η θέση κάθε χαρακτήρα της στήλης καθορίζεται από τον `lineIndex` (ο δείκτης της στήλης) και το `nonogramWidth` (το πλάτος του παζλ). Ο βρόχος ακολουθεί τα εξής βήματα:

- Ξεκινά από τη θέση της στήλης στον πίνακα της συνολικής λύσης (`lineIndex`).
- Προσθέτει το `nonogramWidth` στη θέση, μεταβαίνοντας στον επόμενο χαρακτήρα της ίδιας στήλης.
- Επαναλαμβάνεται έως ότου εξαντληθούν όλοι οι χαρακτήρες της στήλης.

9.5.3.2.2.1.2 Παράδειγμα Υπολογισμού

Έστω ότι το πλάτος του παζλ είναι 4 και η συνολική λύση:

$$\text{solution} = \text{”123412341234”}$$

Αν η στήλη που πρέπει να επεξεργαστεί είναι η δεύτερη (`lineIndex` = 1), ο αλγόριθμος εντοπίζει τους χαρακτήρες ως εξής:

- Στη θέση `1` (2ος χαρακτήρας): `"2"`
- Στη θέση `5` (2ος χαρακτήρας της επόμενης γραμμής): `"2"`
- Στη θέση `9` (2ος χαρακτήρας της τρίτης γραμμής): `"2"`

9.5.3.2.2 Δημιουργία της Τελικής Συμβολοσειράς

Οι χαρακτήρες που εντοπίζονται σε κάθε επανάληψη προστίθενται στη συμβολοσειρά `columnSol` μέσω της πράξης:

$$\text{columnSol} \mathrel{+}= \text{character}$$

Στο τέλος του βρόχου, η `columnSol` περιέχει τους χαρακτήρες της στήλης στη σωστή σειρά.

9.5.3.2.2.1 Παράδειγμα Δημιουργίας

Για το παραπάνω παράδειγμα, η τελική συμβολοσειρά που αντιστοιχεί στη δεύτερη στήλη είναι:

$$\text{”222”}$$

9.5.3.2.3 Τελικό Αποτέλεσμα

Η μέθοδος `getSolutionLine` επιστρέφει τη συμβολοσειρά `columnSol`, που αντιπροσωπεύει την κατάσταση της στήλης. Το αποτέλεσμα είναι κατάλληλο για περαιτέρω επεξεργασία, όπως ο έλεγχος πληρότητας ή η ανάλυση πιθανών λύσεων.

9.5.3.2.3.1 Παράδειγμα Εισόδου και Εξόδου

- **Είσοδος:**
  - `solution`: ”123412341234”
  - `nonogramWidth`: 4

- `lineIndex`: 1 (2η στήλη)

- **Έξοδος:**

$$\text{columnSol} = \text{”222”}$$

9.5.3.2.2.4 Προηγούμενη Υλοποίηση με Χρήση της Characters

Στην αρχική υλοποίηση, ο κώδικας αξιοποιούσε την επέκταση Characters της Flutter για τη διαχείριση της συμβολοσειράς. Η Characters προσέφερε έναν εύχρηστο και αποδοτικό τρόπο πρόσβασης στους χαρακτήρες της λύσης, επιτρέποντας τη διαχείριση τους χωρίς περίπλοκους χειρισμούς. Επέστρεφε άμεσα τους χαρακτήρες της λύσης ως λίστα, με κάθε χαρακτήρα να αποτελεί ξεχωριστό αντικείμενο αυτής. Αυτή η προσέγγιση ήταν εξαιρετικά πρακτική και γρήγορη, καθιστώντας την ιδιαίτερα αποτελεσματική για τον σκοπό της.

Παρά τα πλεονεκτήματά της, η υλοποίηση με χρήση της Characters αντικαταστάθηκε αργότερα. Η μετάβαση έγινε προς τη μέθοδο split, που αποτελεί μέρος της Dart αντί της Flutter. Η λειτουργικότητα της split είναι παρόμοια, αν και απαιτεί επιπλέον κώδικα και προσοχή για να διασφαλιστεί η ακριβής ανάκτηση των δεδομένων.

Η αλλαγή αυτή κρίθηκε απαραίτητη λόγω της χρήσης των isolates, η οποία απαιτεί αποκλειστική εξάρτηση από τη Dart. Περισσότερες λεπτομέρειες σχετικά με τη μετάβαση αυτή θα παρουσιαστούν στο αντίστοιχο τμήμα του εγγράφου που αφορά τη διαχείριση των isolates.

9.5.3.3 Εντοπισμός Μη Συμπληρωμένων Στοιχείων

Η διαδικασία εντοπισμού των μη συμπληρωμένων στοιχείων, δηλαδή των θέσεων των ερωτηματικών (`"?"`) στη λύση, πραγματοποιείται με τρόπο που εξετάζει τη συνολική συμβολοσειρά της λύσης του παζλ και όχι μεμονωμένα κάθε γραμμή ή στήλη. Αυτή η προσέγγιση εξασφαλίζει ότι όλες οι θέσεις των `"?"` εντοπίζονται ανεξάρτητα από τη διάταξή τους στις γραμμές ή τις στήλες.

9.5.3.3.1 Χρήση της `indexed`

Η μέθοδος `getCharIndexesOfQuestionMarks` αξιοποιεί τη λειτουργία `indexed` από τη βιβλιοθήκη Dart, η οποία συνδυάζει κάθε χαρακτήρα της λύσης με τη θέση του. Η `indexed` επιστρέφει μια λίστα με ζεύγη της μορφής `(int, String)`, όπου:

- Το `int` είναι η θέση του χαρακτήρα.
- Το `String` είναι ο ίδιος ο χαρακτήρας.

9.5.3.3.1.1 Παράδειγμα

Για τη συμβολοσειρά:

$$\text{”0?1”}$$

Η `indexed` θα επιστρέψει:

$$[(0, \text{”0”}), (1, \text{”?”}), (2, \text{”1”})]$$

9.5.3.3.2 Χρήση Κανονικών Εκφράσεων (RegEx)

Αφού δημιουργηθεί η παραπάνω λίστα, εφαρμόζεται ένας αλγόριθμος βασισμένος σε κανονική έκφραση (Regular Expression) για να εντοπιστούν οι θέσεις όλων των `"?"` στη συμβολοσειρά. Ο κανονικός τύπος που χρησιμοποιείται

είναι:

```
[0-9]+(?=, \?)
```

9.5.3.3.2.1 Λειτουργία του Κανονικού Τύπου

Ο συγκεκριμένος τύπος λειτουργεί ως εξής:

- Το `[0-9]+` εντοπίζει αριθμούς (δηλαδή τις θέσεις των χαρακτήρων).
- Η προσθήκη του `(?=, \?)` διασφαλίζει ότι αυτοί οι αριθμοί ακολουθούνται από τη συμβολοσειρά `", ?"`.

9.5.3.3.2.2 Παράδειγμα Χρήσης RegEx

Για τη συμβολοσειρά:

”((0, 0), (1, ?), (2, 1))”

Η κανονική έκφραση θα επιστρέψει:

[1]

9.5.3.3.3 Δημιουργία Λίστας Θέσεων

Οι αριθμοί που εντοπίζονται από την κανονική έκφραση εξάγονται και αποθηκεύονται σε μια λίστα ως ακέραιοι. Αυτή η λίστα αντιπροσωπεύει όλες τις θέσεις των `"?"` στη συνολική λύση.

9.5.3.3.3.1 Παράδειγμα Εισόδου και Εξόδου

- **Είσοδος:**
  - `solution`: ”((0, 0), (1, ?), (2, 1))”
- **Έξοδος:**

$$\text{positions} = [1]$$

9.5.3.3.4 Χρησιμότητα Λίστας Θέσεων

Η λίστα με τις θέσεις των `"?"` που επιστρέφει η μέθοδος `getCharIndexesOfQuestionMarks` είναι εξαιρετικά χρήσιμη για την περαιτέρω επεξεργασία από τον αλγόριθμο. Παρέχει τις ακριβείς θέσεις των μη συμπληρωμένων στοιχείων, επιτρέποντας την αποδοτική ανάλυση και τη στοχευμένη ενημέρωση της λύσης.

9.5.4 Αρχικός Έλεγχος Ολοκλήρωσης Γραμμής

9.5.4.1 Εισαγωγή

Ο αρχικός έλεγχος ολοκλήρωσης μιας γραμμής ή στήλης αποτελεί θεμελιώδες βήμα στη διαδικασία επίλυσης ενός παζλ εικονόσταυρου. Στόχος του ελέγχου είναι να επιβεβαιωθεί γρήγορα αν η γραμμή έχει ήδη ολοκληρωθεί, αποφεύγοντας περιττούς υπολογισμούς.

Η μέθοδος συγκρίνει τον αριθμό των συμπληρωμένων κουτιών στη γραμμή με το συνολικό άθροισμα των ενδείξεων (*clues*) της γραμμής. Εάν οι δύο τιμές ταυτίζονται, η γραμμή θεωρείται ολοκληρωμένη και η επεξεργασία της σταματά άμεσα.

9.5.4.2 Υπολογισμός Συμπληρωμένων Κουτιών (*filledBoxes*)

Η αρχική φάση του ελέγχου περιλαμβάνει τον υπολογισμό του αριθμού των συμπληρωμένων κουτιών στη γραμμή. Η μέθοδος `sumFilledBoxes`, που αποτελεί μέρος της επέκτασης `NonoStringExtension`, λειτουργεί ως εξής:

- Διασπά τη συμβολοσειρά σε έναν κατάλογο χαρακτήρων με τη χρήση της `split`.
- Διατρέχει κάθε στοιχείο και ελέγχει αν ο χαρακτήρας δεν είναι `"?"`.
- Προσθέτει το αριθμητικό περιεχόμενο κάθε έγκυρου χαρακτήρα στο συνολικό άθροισμα.

9.5.4.2.1 Παράδειγμα Υπολογισμού Συμπληρωμένων Κουτιών

Αρχική συμβολοσειρά: `"1?10?"`

- Διάσπαση σε κατάλογο: `["1", "?", "1", "0", "?"]`
- Υπολογισμός: $1 + 1 + 0 = 2$
- Αποτέλεσμα: 2

9.5.4.3 Σύγκριση με το Άθροισμα Ενδείξεων (*clues.sum*)

Το συνολικό άθροισμα των ενδείξεων (*clues*) της γραμμής υπολογίζεται με τη χρήση της επέκτασης `NonoIntListExtension`. Η μέθοδος `sum` λειτουργεί ως εξής:

- Διατρέχει όλα τα στοιχεία της λίστας των ενδείξεων.
- Προσθέτει κάθε στοιχείο στο συνολικό άθροισμα.

9.5.4.3.1 Παράδειγμα Υπολογισμού Άθροισματος Ενδείξεων

Λίστα ενδείξεων: `[3, 2]`

- Υπολογισμός: $3 + 2 = 5$
- Αποτέλεσμα: 5

Η σύγκριση πραγματοποιείται με τη συνθήκη:

```
bool isLineCompleted = filledBoxes == clues.sum;
```

9.5.4.4 Αποτέλεσμα της Σύγκρισης

Το αποτέλεσμα της σύγκρισης καθορίζει την κατάσταση της γραμμής:

- Αν το *filledBoxes* ισούται με το *clues.sum*, η γραμμή θεωρείται ολοκληρωμένη και περαιτέρω επεξεργασία παραλείπεται.
- Αν δεν ισχύει η συνθήκη, η γραμμή απαιτεί περαιτέρω επεξεργασία.

Αυτή η προσέγγιση μειώνει την περιττή εργασία, επιταχύνει την επίλυση του παζλ και εξοικονομεί υπολογιστικούς πόρους.

9.5.4.5 Παράδειγμα Εφαρμογής

Ας υποθέσουμε την παρακάτω γραμμή:

- Αρχική λύση: `"1 ? 1 1 0 1 ?"`
- Ενδείξεις: `[1, 2, 1]`

9.5.4.5.1 Βήματα Υπολογισμού

- Υπολογισμός *filledBoxes*:
$$1 + 1 + 1 + 0 + 1 = 4$$
- Υπολογισμός *clues.sum*:
$$1 + 2 + 1 = 4$$
- Σύγκριση:
$$4 == 4 \implies \text{Η γραμμή θεωρείται ολοκληρωμένη.}$$

9.5.4.6 Σημασία της Διαδικασίας

Η στρατηγική αυτή βελτιώνει την αποδοτικότητα του αλγορίθμου με τους εξής τρόπους:

- Ελαχιστοποιεί τους περιττούς υπολογισμούς για ήδη ολοκληρωμένες γραμμές.
- Επικεντρώνει την επεξεργασία στις γραμμές που απαιτούν ανάλυση.
- Ενισχύει τη διαφάνεια του αλγορίθμου μέσω σαφών και δομημένων κριτηρίων ολοκλήρωσης.

Συνολικά, ο αρχικός έλεγχος ολοκλήρωσης διαδραματίζει καθοριστικό ρόλο στην επίλυση μεγάλων ή πολύπλοκων παζλ, συμβάλλοντας στη μείωση του συνολικού χρόνου επεξεργασίας.

9.5.5 Επεξεργασία ανάλογα με την κατάσταση της γραμμής

9.5.5.1 Επεξεργασία ολοκληρωμένης γραμμής

9.5.5.1.1 Εισαγωγή

Η διαδικασία επεξεργασίας μιας ολοκληρωμένης γραμμής στο παζλ εικονόσταυρου ξεκινά όταν διαπιστωθεί ότι ο αριθμός των ήδη συμπληρωμένων στοιχείων στη γραμμή ή τη στήλη ισούται με το άθροισμα των ενδείξεων (*clues*) της γραμμής. Σε αυτή την περίπτωση, ο αλγόριθμος προχωρά στην εξάλειψη των μη συμπληρωμένων κουτιών (`"?"`) για την αποφυγή περιττών υπολογισμών.

Η επαλήθευση πραγματοποιείται μέσω της μεταβλητής `isLineCompleted`, η οποία συγκρίνει τον αριθμό των συμπληρωμένων κουτιών (*filledBoxes*) με το συνολικό άθροισμα των ενδείξεων (*clues.sum*) της γραμμής.

9.5.5.1.2 Επιβεβαίωση Υπολοίπων Κουτιών

Μετά τη διαπίστωση ότι η γραμμή είναι συμπληρωμένη, ο αλγόριθμος ελέγχει αν υπάρχουν υπόλοιπα κενά κουτιά (`"?"`) χρησιμοποιώντας:

```
initialSolution.split('').toList().contains('?')
```

Εάν εντοπιστεί ο χαρακτήρας "?", ο αλγόριθμος διαγράφει τα κενά κουτιά και ενημερώνει τη γραμμή. Σε διαφορετική περίπτωση, η μέθοδος τερματίζει, επιστρέφοντας τα ήδη συλλεχθέντα δεδομένα.

9.5.5.1.3 Εντοπισμός και Εξάλειψη Μη Συμπληρωμένων Κουτιών

Η ενημέρωση της λύσης, μέσω της διαγραφής μη συμπληρωμένων κουτιών, πραγματοποιείται με τη χρήση της μεθόδου `getFilledInSolution`, η οποία αντικαθιστά τα κενά κουτιά με την τιμή "0".

9.5.5.1.3.1 Λεπτομερής Περιγραφή της `getFilledInSolution`

9.5.5.1.3.1.1 Είσοδοι και Αρχικοποίηση

Η μέθοδος `getFilledInSolution` δέχεται τις εξής παραμέτρους:

- `currentSolution`: Η τρέχουσα κατάσταση της λύσης (συμβολοσειρά).
- `lineIndex`: Ο δείκτης της γραμμής ή στήλης που επεξεργάζεται.
- `lineType`: Ο τύπος του άξονα (`NonoAxis.row` ή `NonoAxis.column`).
- `nonogramWidth`: Το πλάτος του παζλ.
- `charIndexes`: Λίστα με τις θέσεις των χαρακτήρων προς αντικατάσταση.
- `clueKey`: Τιμή αντικατάστασης (`0` για διαγραφή, `1` για συμπλήρωση).

Αρχικά, δημιουργείται η λίστα `newFilledBoxes` για να αποθηκεύσει τις θέσεις των ενημερωμένων κουτιών.

9.5.5.1.3.1.2 Υπολογισμός Θέσης Χαρακτήρα (getSolutionPosition)

Για κάθε θέση στη λίστα `charIndexes`, υπολογίζεται η ακριβής θέση στη συμβολοσειρά της λύσης:

- Αν ο άξονας είναι γραμμή (NonoAxis.row), η θέση υπολογίζεται ως:

$$\texttt{tempPos} = \texttt{lineIndex} \times \texttt{nonogramWidth} + \texttt{charIndex}$$

- Αν ο άξονας είναι στήλη (NonoAxis.column), η θέση υπολογίζεται ως:

$$\texttt{tempPos} = \texttt{charIndex} \times \texttt{nonogramWidth} + \texttt{charIndex}$$

Με αυτόν τον τρόπο, η getSolutionPosition διασφαλίζει τη σωστή αντιστοίχιση μεταξύ των γραμμών ή στηλών και της συνολικής λύσης.

9.5.5.1.3.1.3 Αντικατάσταση Χαρακτήρων

Αφού υπολογιστεί η θέση, η μέθοδος αντικαθιστά τον χαρακτήρα στη συγκεκριμένη θέση της συμβολοσειράς με την τιμή που ορίζεται από την παράμετρο `clueKey`:

- `0`: Υποδηλώνει ότι το κουτί διαγράφεται.
- `1`: Υποδηλώνει ότι το κουτί συμπληρώνεται.

Αυτό επιτυγχάνεται με τη χρήση της παρακάτω μεθόδου:

```
fullUpdatedSolution =
    '${fullUpdatedSolution.substring(0, tempPos)}${clueKey == 0 ? '0' :
    '1'}${fullUpdatedSolution.substring(tempPos + 1)}';
```

Παράλληλα, η θέση αυτή καταγράφεται στη λίστα `newFilledBoxes`, ώστε να ενημερωθεί η λίστα (σωρός) για περαιτέρω επεξεργασία.

9.5.5.1.3.1.4 Επιστροφή Αποτελέσματος

Στο τέλος, η μέθοδος επιστρέφει έναν χάρτη (`map`) με δύο βασικά στοιχεία:

- `fullUpdatedSolution:` Τη νέα, ενημερωμένη συμβολοσειρά της λύσης.
- `newFilledBoxes:` Τη λίστα με τις θέσεις των νέων κουτιών που συμπληρώθηκαν ή διαγράφηκαν.

9.5.5.1.3.2 Ανανέωση Ολοκληρωμένης Γραμμής (`getFilledInSolution`)

Κατά την επεξεργασία μιας γραμμής που έχει χαρακτηριστεί ως πλήρως συμπληρωμένη, η `getFilledInSolution` καλείται για να αντικαταστήσει όλα τα μη συμπληρωμένα κουτιά (`"?"`) με την τιμή `0`, υποδεικνύοντας ότι τα κουτιά αυτά διαγράφηκαν.

Η νέα συμβολοσειρά που επιστρέφει η μέθοδος αποθηκεύεται ως η ενημερωμένη λύση. Παράλληλα, οι θέσεις των κουτιών που ενημερώθηκαν καταγράφονται και χρησιμοποιούνται για την ανανέωση της λίστας (σωρού).

Η χρήση της `getFilledInSolution` συμβάλλει στην αποφυγή επαναλαμβανόμενων υπολογισμών και εξασφαλίζει ότι η επεξεργασία επικεντρώνεται μόνο στις απαραίτητες γραμμές και στήλες, διασφαλίζοντας έτσι την αποδοτική λειτουργία του αλγορίθμου.

9.5.5.1.3.2.1 Παράδειγμα

- Αρχική συμβολοσειρά: `"1?1?"`
- `charIndexes: [1, 3]`
- Τελική συμβολοσειρά: `"1010"`

9.5.5.2 Επεξεργασία μη ολοκληρωμένης γραμμής

Η επεξεργασία μιας μη ολοκληρωμένης γραμμής αποτελεί ένα κρίσιμο στάδιο στη διαδικασία επίλυσης ενός παζλ εικονόσταυρου. Στο στάδιο αυτό, ο αλγόριθμος αναλύει τη γραμμή σε βάθος, με στόχο να εξάγει νέες πληροφορίες που θα οδηγήσουν στη συμπλήρωση ή τη διαγραφή κουτιών. Η διαδικασία περιλαμβάνει:

- Υπολογισμό όλων των πιθανών λύσεων.
- Εντοπισμό των ακραίων λύσεων.
- Σύγκριση των ακραίων λύσεων.
- Ενημέρωση της κατάστασης της λύσης.
- Εισαγωγή νέων στοιχείων στη λίστα (σωρό) για περαιτέρω επεξεργασία.

Σημειώνεται ότι οι μέθοδοι `_getAllLinePossibleSolutions`, `_getSideMostSolution`, και `_getSideMostSolutionsMatches` αναλύονται λεπτομερώς σε επόμενα τμήματα.

9.5.5.2.1 Υπολογισμός Όλων των Πιθανών Λύσεων

Το πρώτο βήμα της διαδικασίας είναι ο υπολογισμός όλων των πιθανών λύσεων με τη μέθοδο `_getAllLinePossibleSolutions`. Η μέθοδος λαμβάνει ως είσοδο:

- Τις ενδείξεις (`clues`).
- Την αρχική κατάσταση της γραμμής.
- Τα δεδομένα της τρέχουσας λύσης.
- Τις ρυθμίσεις του αλγορίθμου.

Η έξοδος της μεθόδου είναι μια λίστα από λίστες, όπου κάθε υπολίστα αντιστοιχεί σε ένα κουτί της γραμμής. Τα στοιχεία της υπολίστας αναπαριστούν τις ενδείξεις που μπορούν να τοποθετηθούν σε αυτή τη θέση, διευκολύνοντας τη σύγκριση και την εξαγωγή ασφαλών συμπερασμάτων.

9.5.5.2.2 Εντοπισμός Ακραίων Λύσεων

Μετά τον υπολογισμό των πιθανών λύσεων, η μέθοδος `_getSideMostSolution` καλείται δύο φορές:

- Για την **Αρχική Ακραία Λύση** (`Starting Most Solution`), εντοπίζοντας τη λύση που τοποθετεί τις ενδείξεις όσο πιο αριστερά γίνεται στη γραμμή (ή πάνω, για στήλες).
- Για την **Τελική Ακραία Λύση** (`Ending Most Solution`), εντοπίζοντας τη λύση που τοποθετεί τις ενδείξεις όσο πιο δεξιά γίνεται στη γραμμή (ή κάτω, για στήλες).

Η έξοδος της διαδικασίας είναι μια λίστα συμβολοσειρών που υποδεικνύουν τις ενδείξεις που τοποθετούνται σε κάθε θέση της γραμμής.

9.5.5.2.3 Σύγκριση Ακραίων Λύσεων

Η μέθοδος `_getSideMostSolutionsMatches` συγκρίνει τις δύο ακραίες λύσεις. Αν μια ένδειξη βρίσκεται στην ίδια θέση και στις δύο λύσεις, αυτή θεωρείται βέβαιη και τοποθετείται στη γραμμή. Το αποτέλεσμα είναι ένας χάρτης (`map`) με τις θέσεις των κουτιών που μπορούν να συμπληρωθούν με σιγουριά.

9.5.5.2.4 Ενημέρωση της Κατάστασης Λύσης

Με τις βέβαιες ενδείξεις διαθέσιμες, η μέθοδος `getFilledInSolution` χρησιμοποιείται για να ενημερώσει τη λύση:

- Υπολογίζει τις θέσεις των κουτιών που πρέπει να αλλάξουν.
- Ενημερώνει τη συμβολοσειρά της λύσης αντικαθιστώντας τους χαρακτήρες στις θέσεις με:
  - `0` για διαγραφή.
  - `1` για συμπλήρωση.
- Επιστρέφει την ενημερωμένη λύση και μια λίστα με τις θέσεις των κουτιών που τροποποιήθηκαν.

9.5.5.2.5 Προσθήκη Στοιχείων στη Λίστα (Σωρό)

Οι νέες πληροφορίες προστίθενται στη λίστα (σωρό) με τη μέθοδο `getNewStackElements`, η οποία εντοπίζει ποιες γραμμές ή στήλες επηρεάζονται από τις αλλαγές. Αυτό εξασφαλίζει ότι ο αλγόριθμος επικεντρώνεται στις κρίσιμες περιοχές του παζλ.

9.5.5.2.6 Επιστροφή Ενημερωμένων Αποτελεσμάτων

Η διαδικασία ολοκληρώνεται με την επιστροφή ενός αντικειμένου `IsolateOutput`, που περιλαμβάνει:

- Την ενημερωμένη κατάσταση της λύσης.
- Τα βήματα λύσης, με λεπτομέρειες για τις αλλαγές.
- Την ενημερωμένη λίστα (σωρό) για περαιτέρω επεξεργασία.
- Στατιστικά δεδομένα, όπως τον αριθμό των γραμμών που επεξεργάστηκαν.

Αυτός ο μηχανισμός διασφαλίζει ότι η λύση του παζλ προχωρά με αποδοτικό και συστηματικό τρόπο.

9.5.6 Επίλογος

Η μέθοδος `loopSides` αποτελεί τον πυρήνα του αλγορίθμου επίλυσης γραμμών και στηλών σε παζλ εικονόσταυρου, προσφέροντας μια ολοκληρωμένη προσέγγιση για την ανάλυση και τη διαχείριση των δεδομένων κάθε γραμμής ή στήλης. Μέσα από τις επιμέρους διαδικασίες της, η μέθοδος διασφαλίζει ότι κάθε γραμμή αντιμετωπίζεται με τον πιο αποδοτικό τρόπο, είτε πρόκειται για πλήρως ολοκληρωμένες γραμμές είτε για γραμμές που απαιτούν περαιτέρω επεξεργασία.

9.5.6.1 Βασική Δομή και Λειτουργία

Η δομή της μεθόδου βασίζεται σε έναν συστηματικό έλεγχο της κατάστασης της γραμμής, ο οποίος διαχωρίζει τις ολοκληρωμένες από τις μη ολοκληρωμένες γραμμές:

- **Ολοκληρωμένες Γραμμές:** Οι υπολειπόμενες ενέργειες επικεντρώνονται στη διαγραφή των εναπομεινάντων κενών κουτιών και στην ενημέρωση της λύσης.
- **Μη Ολοκληρωμένες Γραμμές:** Ο αλγόριθμος προχωρά σε βαθύτερη ανάλυση, υπολογίζοντας πιθανές λύσεις, εντοπίζοντας ακραίες λύσεις, συγκρίνοντας αυτές για την εξαγωγή βέβαιων αποτελεσμάτων και ανανεώνοντας τη λίστα (σωρό) με τις επηρεασμένες γραμμές και στήλες.

9.5.6.2 Σύγχρονες Μέθοδοι και Εργαλεία

Η ενσωμάτωση μεθόδων όπως οι `_getAllLinePossibleSolutions`, `_getSideMostSolution`, `_getSideMostSolutionsMatches`, και `getFilledInSolution` επιτρέπει στον αλγόριθμο να διαχειρίζεται πολύπλοκες καταστάσεις με ευελιξία και ακρίβεια. Παρόλο που αυτές οι μέθοδοι αναφέρονται εν συντομία εδώ, η πλήρης τους ανάλυση πραγματοποιείται στο επόμενο κεφάλαιο.

9.5.6.3 Στρατηγική Αποδοτικότητας

Η μεθοδολογία της `loopSides` προσφέρει μια ισορροπημένη συνάρτηση μεταξύ ταχύτητας και ακρίβειας, προσαρμοσμένη στις απαιτήσεις ενός παζλ εικονόσταυρου. Μέσω της βελτιστοποιημένης στρατηγικής ελέγχου και επεξεργασίας, εξασφαλίζεται:

- Αποδοτική επίλυση ακόμα και για μεγάλα ή πολύπλοκα παζλ.
- Μείωση περιττής επεξεργασίας.
- Επιτάχυνση της συνολικής διαδικασίας.

9.5.6.4 Φιλοσοφία Σχεδίασης

Η μέθοδος `loopSides` δεν αποτελεί μόνο έναν τεχνικό μηχανισμό για την επίλυση, αλλά και μια φιλοσοφία στρατηγικής επίλυσης, η οποία δίνει έμφαση:

- Στη στοχευμένη χρήση των υπολογιστικών πόρων.
- Στην αποφυγή άσκοπων βημάτων.
- Στην ενίσχυση της συνολικής αποδοτικότητας του αλγορίθμου.

Αυτή η προσέγγιση θέτει τις βάσεις για την ανάπτυξη ακόμα πιο προηγμένων μεθόδων επίλυσης στο μέλλον.

9.5.6.5 Λειτουργίες της `loopSides`

Πιο συνοπτικά και μαζεμένα, η `loopSides` τρέχει τις παρακάτω λειτουργίες:

1. **Ανάκτηση δεδομένων γραμμής:**
    - Ανάκτηση της αρχικής κατάστασης: Με τη μέθοδο `getSolutionLine`, τραβάει την κατάσταση των κουτιών της γραμμής ή στήλης που θα του δοθεί.
    - Εύρεση θέσεων μη συμπληρωμένων κουτιών: Τα ”?” (ερωτηματικά) στη λύση, που αντιπροσωπεύουν μη συμπληρωμένα στοιχεία, προσδιορίζονται μέσω της `getCharIndexesOfQuestionMarks`.
2. **Αρχικός έλεγχος ολοκλήρωσης γραμμής:**
    - Εξετάζει αν η γραμμή έχει ήδη ολοκληρωθεί με έναν απλό έλεγχο καταμέτρησης συμπληρωμένων στοιχείων (`clues`).
3. **Επεξεργασία ανάλογα με την κατάσταση της γραμμής:**
    - **Ολοκληρωμένη γραμμή:**
        - Εφόσον υπάρχουν ακόμα κενά κουτιά (”?”), αυτά διαγράφονται, και η γραμμή ενημερώνεται.
        - Το αποτέλεσμα αποθηκεύεται και η λίστα (σωρός) ενημερώνεται με νέες επιρροές από τη διαγραφή αυτών των κουτιών.
        - Ο αλγόριθμος τερματίζει και επιστρέφει τα αποτελέσματα της γραμμής.
    - **Μη ολοκληρωμένη γραμμή:**
        - Υπολογίζονται όλες οι πιθανές λύσεις μέσω της `_getAllLinePossibleSolutions`.

- Γίνεται σύγκριση μεταξύ της αρχικής και της τελικής κατάστασης για να προσδιοριστούν σίγουρα κουτιά που πρέπει να γεμίσουν ή να διαγραφούν.
- Επεξεργασία στήλης αν υπάρχουν κοινά σημεία ανάμεσα στις δύο λύσεις.
  * Εύρεση όλων των κουτιών που χρειάζονται να γεμίσουν ανα στοιχείο.
  * Γίνεται έλεγχος του αν η γραμμή είναι συμπληρωμένη, έτσι ώστε να σβήσει τα υπόλοιπα κενά.
  * Υπολογίζει ποιες γραμμές ή στήλες επηρεάζονται, ώστε να επιστραφουν για να προστεθουν στη λίστα (σωρό) για να επανεξεταστούν.
  * Η ενημερωμένη λύση αποθηκεύεται, ενώ νέα δεδομένα προστίθενται στη λίστα (σωρό) για περαιτέρω επεξεργασία.
- Η ενημερωμένη λύση αποθηκεύεται.

Η `loopSides` διασφαλίζει τη συστηματική και αποδοτική διαχείριση κάθε γραμμής, ενισχύοντας την ακρίβεια και την ταχύτητα του αλγορίθμου επίλυσης.

9.6 Μέθοδοι Υπολογισμού Όλων των Πιθανών Λύσεων Γραμμής ή Στήλης

Η μέθοδος `_getAllLinePossibleSolutions` αποτελεί την κύρια διαδικασία για την εύρεση όλων των πιθανών λύσεων μιας γραμμής ή στήλης σε ένα παζλ εικονόσταυρου. Αξιοποιεί τις *ενδείξεις (clues)* και την τρέχουσα κατάσταση της γραμμής για να καθορίσει ποιες θέσεις μπορούν να συμπληρωθούν, παρέχοντας έτσι έναν πλήρη χάρτη πιθανών λύσεων για κάθε χαρακτήρα της γραμμής.

9.6.1 Σημαντικές Πληροφορίες Πριν το Διάβασμα των Παρακάτω Ενοτήτων

Στις μεθόδους αυτού του κεφαλαίου υιοθετήθηκαν συγκεκριμένες παραδοχές, οι οποίες αφορούν τη διαδικασία αναπαράστασης και κωδικοποίησης των ενδείξεων. Αυτές οι πληροφορίες είναι κρίσιμες για την κατανόηση των επόμενων ενοτήτων.

9.6.1.1 Αναπαράσταση Ενδείξεων Μέσω Κωδικοποίησης (`clueIndex + 2`)

Η διαδικασία εύρεσης λύσεων απαιτεί έναν τρόπο καταγραφής των στοιχείων, ώστε να μπορούν να ενσωματωθούν στην τελική λύση ως συμπληρωμένα κουτάκια. Η αναπαράσταση των στοιχείων μέσω *κωδικοποίησης* επιλύει τα προβλήματα που προκύπτουν από την άμεση χρήση των δεικτών (indexes).

9.6.1.1.1 Προβλήματα Άμεσης Χρήσης των Δεικτών

Η χρήση των αριθμών 0 και 1, που έχουν ειδική σημασία στον αλγόριθμο, αποδείχθηκε μη εφαρμόσιμη για την αναπαράσταση των στοιχείων:

- Το **0** σηματοδοτεί ότι τα κουτάκια είναι *κενά*, δηλαδή κανένα στοιχείο δεν μπορεί να τοποθετηθεί εκεί (οπτικά διαγράφονται με `X`).
- Το **1** υποδεικνύει ότι το κουτί έχει *γεμίσει*, χωρίς ωστόσο να προσδιορίζεται ποιο στοιχείο ευθύνεται για αυτό το γέμισμα.

Η χρήση των δεικτών για την αναπαράσταση των στοιχείων απαιτούσε έναν ξεχωριστό αλγόριθμο για την ανάκτηση του αρχικού στοιχείου, ο οποίος θα έπρεπε να αξιολογηθεί για την αποδοτικότητά του.

9.6.1.1.2 Εναλλακτική Λύση: Κωδικοποίηση Δεικτών με `clueIndex + 2`

Για την αντιμετώπιση του παραπάνω προβλήματος, υιοθετήθηκε μια απλή αλλά αποδοτική κωδικοποίηση:

- Προστίθεται ο αριθμός 2 στον δείκτη (`index`) κάθε στοιχείου της λίστας ενδείξεων.
- Τα **indexes ξεκινούν από τον αριθμό 2**, αποφεύγοντας έτσι τη σύγκρουση με τα 0 και 1.
- Ο αλγόριθμος παραμένει *αναστρέψιμος*, επιτρέποντας την εύκολη ανάκτηση του αρχικού `index`, όταν αυτό χρειαστεί.

9.6.1.1.3 Πλεονεκτήματα της Κωδικοποίησης

Η χρήση αυτής της κωδικοποίησης παρέχει τα εξής πλεονεκτήματα:

- **Αποφυγή Συγκρούσεων:** Τα 0 και 1 διατηρούν τις ειδικές τους σημασίες στον αλγόριθμο.
- **Απλότητα Υλοποίησης:** Η κωδικοποίηση είναι εύκολη να ενσωματωθεί και να χρησιμοποιηθεί.
- **Αναστρεψιμότητα:** Η επιστροφή στον αρχικό δείκτη (`index`) είναι απλή και αποδοτική.

Η παραπάνω στρατηγική καθιστά τη διαδικασία εύρεσης λύσεων σαφή και αποδοτική, επιτρέποντας την ομαλή λειτουργία του αλγορίθμου και την αποφυγή περιττής πολυπλοκότητας.

9.6.2 Κύρια Μέθοδος Υπολογισμού Όλων των Πιθανών Λύσεων (`_getAllLinePossibleSolutions`)

9.6.2.1 Εισαγωγή

Η μέθοδος `_getAllLinePossibleSolutions` υπολογίζει όλες τις πιθανές λύσεις για μια συγκεκριμένη γραμμή (οριζόντια ή κάθετη) ενός παζλ εικονόσταυρου. Χρησιμοποιεί τις *ενδείξεις (clues)* και την τρέχουσα κατάσταση της γραμμής (`solution string`) για να καθορίσει ποιες θέσεις μπορούν να γεμίσουν και ποιες παραμένουν κενές. Δημιουργεί μια λεπτομερή αναπαράσταση όλων των πιθανών καταστάσεων της γραμμής, η οποία είναι απαραίτητη για την πρόοδο του αλγορίθμου.

9.6.2.2 Σύνδεση με Άλλες Μεθόδους

Η μέθοδος συνεργάζεται με τις `_canCluesFit` και `_doOtherCluesFit`, που εξετάζονται στις ενότητες *Κεφάλαιο 9.7.2* και *Κεφάλαιο 9.7.3*, αντίστοιχα. Επίσης, χρησιμοποιεί την κωδικοποίηση `clueIndex + 2`, όπως εξηγήθηκε στο *Κεφάλαιο 9.6.1.1*, για αποφυγή συγκρούσεων με τις βασικές τιμές 0 και 1.

9.6.2.3 Αρχικοποίηση των Πιθανών Λύσεων

9.6.2.3.1 Δημιουργία Δομής Δεδομένων

Το πρώτο βήμα είναι η δημιουργία μιας δομής δεδομένων που αποθηκεύει όλες τις πιθανές λύσεις για κάθε θέση της γραμμής. Αυτή η δομή αρχικοποιείται ως εξής:

```
final List<List<String>> possibleSolutions =
    Iterable<List<String>>.generate(line.length, (_) => <String>[]).toList();
```

Η εντολή δημιουργεί μια λίστα με τόσα στοιχεία όσα τα κουτιά της γραμμής. Κάθε στοιχείο είναι μια κενή λίστα που θα γεμίσει με τις δυνατές τιμές.

9.6.2.3.1.1 Πλεονεκτήματα

Αυτή η δομή δεδομένων διασφαλίζει βέλτιστη χρήση της μνήμης και ταχύτητα πρόσβασης στις πιθανές λύσεις, διευκολύνοντας τη δυναμική ενημέρωση κατά την επεξεργασία.

9.6.2.3.1.2 Επεκτάσεις

Η δομή είναι αποδοτική για γραμμές μικρού μεγέθους. Σε μεγαλύτερες γραμμές, απαιτείται προσεκτική διαχείριση της μνήμης για να αποφευχθεί η εκθετική αύξηση της πολυπλοκότητας.

9.6.2.4 Επαναληπτική Διαδικασία

9.6.2.4.1 Λειτουργία Βασικού Αλγορίθμου

Η μέθοδος χρησιμοποιεί έναν επαναληπτικό αλγόριθμο οργανωμένο σε δύο εμφωλευμένους βρόχους `for`. Ο εξωτερικός βρόχος διατρέχει τις ενδείξεις (*clues*), ενώ ο εσωτερικός εξετάζει πιθανές θέσεις για κάθε ένδειξη. Ο σκοπός αυτής της διαδικασίας είναι να ελεγχθεί αν κάθε ένδειξη μπορεί να τοποθετηθεί σε κάθε πιθανή θέση της γραμμής ή της στήλης και να αποθηκευτεί το αποτέλεσμα για περαιτέρω επεξεργασία.

9.6.2.4.1.1 Αρχική Υλοποίηση

Η αρχική υλοποίηση εκτελούσε πλήρη έλεγχο όλων των θέσεων της γραμμής, από την πρώτη (index 0) έως την τελευταία. Αν και εξαντλητική, ήταν υπολογιστικά βαριά για μεγαλύτερα παζλ.

9.6.2.4.1.2 Βελτίωση με Υπολογισμούς Ορίων

Με την εισαγωγή αλγορίθμου διακρίσεων, ο αλγόριθμος υπολογίζει ελάχιστα και μέγιστα σημεία εκκίνησης για κάθε ένδειξη. Αυτό περιγράφεται λεπτομερώς στην επόμενη ενότητα και μειώνει δραστικά τους περιττούς ελέγχους.

9.6.2.5 Υπολογισμός Ελάχιστου και Μέγιστου Σημείου Εκκίνησης

Ο υπολογισμός του **ελάχιστου** και **μέγιστου σημείου εκκίνησης** στοχεύει στη μείωση του χρόνου υπολογισμού στον αλγόριθμο, περιορίζοντας τις θέσεις που πρέπει να εξεταστούν για την τοποθέτηση των ενδείξεων. Παρόλο που η διαδικασία αυτή δεν επιλύει το παζλ άμεσα, συμβάλλει στη βελτίωση της συνολικής αποδοτικότητας αποφεύγοντας επαναλαμβανόμενους και περιττούς υπολογισμούς.

9.6.2.5.1 Θεωρητική Βάση

Κάθε ένδειξη μπορεί να αρχίσει να τοποθετείται από το **ελάχιστο σημείο** που καθορίζεται από τις προηγούμενες ενδείξεις της γραμμής ή στήλης και τα απαραίτητα κενά ανάμεσά τους. Παρομοίως, το **μέγιστο σημείο** περιορίζεται από το συνολικό πλάτος της γραμμής ή στήλης και τα στοιχεία που ακολουθούν.

9.6.2.5.1.1 Παράδειγμα Υπολογισμού Ελαχίστων και Μεγίστων Σημείων

Έστω η λίστα ενδείξεων $[3, 2, 3, 1, 4]$ και η γραμμή έχει πλάτος 20. Εξετάζεται η τρίτη ένδειξη ($index = 2$):

- **Ελάχιστο Σημείο Εκκίνησης:** Υπολογίζεται ως το άθροισμα των προηγούμενων ενδείξεων και των κενών:

$$\text{Ελάχιστο} = (3 + 2) + 2 = 7 \quad \text{(index 6)}.$$

- **Μέγιστο Σημείο Εκκίνησης:** Προκύπτει αφαιρώντας από το πλάτος το άθροισμα των επόμενων στοιχείων και των κενών:

$$\text{Μέγιστο} = 20 - (1 + 4) - 2 = 13 \quad \text{(index 12)}.$$

9.6.2.5.1.1.1 Σημαντική Μείωση Υπολογισμών

Πριν την εφαρμογή αυτής της βελτιστοποίησης, ο αλγόριθμος επαναλάμβανε ελέγχους σε όλες τις θέσεις, από $index = 0$ έως $index = 19$ (20 συνολικές επαναλήψεις). Με την εφαρμογή των ελαχίστων και μεγίστων σημείων, οι έλεγχοι περιορίζονται από $index = 6$ έως $index = 12$ (μόλις 6 επαναλήψεις), μειώνοντας τον αριθμό των ελέγχων κατά περισσότερο από 50%.

9.6.2.5.1.2 Επιπτώσεις στη Βελτιστοποίηση

Αυτή η στρατηγική εξοικονομεί σημαντικό χρόνο, απορρίπτοντας από νωρίς θέσεις που δεν πληρούν τις προϋποθέσεις τοποθέτησης μιας ένδειξης. Η εξοικονόμηση χρόνου υπολογισμού είναι κρίσιμη σε μεγάλα ή σύνθετα παζλ, όπου οι περιττές επαναλήψεις ενδέχεται να αυξήσουν δραματικά την πολυπλοκότητα.

9.6.2.5.1.2.1 Αναφορές σε Επιμέρους Συναρτήσεις

Οι υλοποιήσεις των συναρτήσεων που πραγματοποιούν αυτούς τους υπολογισμούς περιγράφονται λεπτομερώς στις ενότητες:

- Κεφάλαιο 9.9.7.3.2 `minStartingPoint`
- Κεφάλαιο 9.9.7.3.3 `maxStartingPoint`.

Αυτές οι συναρτήσεις εξασφαλίζουν ακρίβεια στους υπολογισμούς και ενσωματώνονται αποτελεσματικά στον συνολικό αλγόριθμο επίλυσης.

9.6.2.6 Έλεγχος αν η Ένδειξη μπορεί να τοποθετηθεί

Στο στάδιο αυτό, ο στόχος είναι να ελεγχθεί αν η ένδειξη που βρίσκεται στη διαδικασία επαναλήψεων, όπως περιγράφεται στην ενότητα *Κεφάλαιο 9.6.2.4 Επαναληπτική Διαδικασία*, μπορεί να τοποθετηθεί στη συγκεκριμένη θέση που εξετάζεται. Ο έλεγχος αυτός πραγματοποιείται σε δύο διαδοχικά βήματα: πρώτα γίνεται αναζήτηση των ήδη αποθηκευμένων δεδομένων στη μνήμη cache, και στη συνέχεια, αν δεν υπάρχει σχετική πληροφορία, πραγματοποιείται υπολογισμός μέσω της συνάρτησης `_canCluesFit`.

9.6.2.6.1 Έλεγχος στη Μνήμη Cache

Ο πρώτος έλεγχος του αλγορίθμου είναι η αναζήτηση στη μνήμη *cache*, ώστε να διαπιστωθεί αν έχει ήδη γίνει υπολογισμός για το αν η συγκεκριμένη ένδειξη (*clue*) μπορεί να τοποθετηθεί στη θέση που εξετάζεται. Ο έλεγχος αυτός εκτελείται μέσω της πρόσβασης στο αντικείμενο `cachedBoxSolutions`, το οποίο περιέχει τις ήδη γνωστές απαντήσεις.

```
final bool? cache = settings.keepCacheData
    ? output.cachedBoxSolutions['$clues,$clueIndex,$line,$charIndex']
```

```
    : null;
final bool isInCache = cache != null;
```

Ο παραπάνω έλεγχος επιβεβαιώνει αν υπάρχει αποθηκευμένη τιμή για τη συγκεκριμένη ένδειξη, γραμμή ή στήλη και θέση. Αν βρεθεί τιμή στη μνήμη *cache* (`isInCache == true`), ο αλγόριθμος παρακάμπτει την επόμενη διαδικασία υπολογισμού και χρησιμοποιεί το ήδη αποθηκευμένο αποτέλεσμα. Αυτό μειώνει τον χρόνο εκτέλεσης, καθώς αποφεύγονται περιττοί επαναληπτικοί υπολογισμοί.

Περισσότερες πληροφορίες για τη διαχείριση της μνήμης *cache* και τον τρόπο αποθήκευσης των δεδομένων υπάρχουν στην ενότητα *Κεφάλαιο 9.8.6 Διαχείριση Μνήμης Cache (`updateCachedBoxSolutions`)*.

9.6.2.6.2 Έλεγχος Τοποθέτησης Ενδείξεων (`_canCluesFit`)

Αν δεν υπάρχει σχετική πληροφορία στη μνήμη *cache*, ο αλγόριθμος προχωρά στον υπολογισμό του αν η ένδειξη μπορεί να τοποθετηθεί στη θέση που εξετάζεται. Αυτό επιτυγχάνεται μέσω της συνάρτησης `_canCluesFit`, η οποία υλοποιεί μια σειρά ελέγχων και επαληθεύσεων. Η συνάρτηση αυτή καλεί, όπου είναι απαραίτητο, και άλλες μεθόδους, όπως τη `_doOtherCluesFit`, προκειμένου να διαπιστώσει αν οι υπόλοιπες ενδείξεις της γραμμής ή στήλης είναι συμβατές με τη συγκεκριμένη τοποθέτηση.

```
final List<List<String>> possibleSolutions =
    Iterable<List<String>>.generate(line.length, (_) => <String>[]).toList();
```

Η `_canCluesFit` αναλύει αν οι ενδείξεις της λίστας μπορούν να ευθυγραμμιστούν στη γραμμή ή τη στήλη, λαμβάνοντας υπόψη τη θέση εκκίνησης που δόθηκε από την επαναληπτική διαδικασία. Αν το αποτέλεσμα είναι `true`, σημαίνει ότι η ένδειξη μπορεί να τοποθετηθεί στη συγκεκριμένη θέση. Αν είναι `false`, η θέση απορρίπτεται για την ένδειξη αυτή.

Για λεπτομέρειες σχετικά με τη λειτουργία της `_canCluesFit`, μπορείτε να ανατρέξετε στο Κεφάλαιο 9.7.2 `_canCluesFit`.

9.6.2.6.3 Τελική Απόφαση για την Τοποθέτηση

Στο τέλος αυτής της διαδικασίας, είτε μέσω της μνήμης *cache* είτε μέσω του υπολογισμού, λαμβάνεται απόφαση για το αν η ένδειξη μπορεί να τοποθετηθεί στη θέση που εξετάζεται. Αν το αποτέλεσμα είναι θετικό (`true`), ο αλγόριθμος προχωρά στην ενημέρωση της λίστας πιθανών λύσεων για τη γραμμή ή τη στήλη, προσθέτοντας τις κατάλληλες πληροφορίες.

Αυτή η διαδικασία συνεχίζεται για κάθε ένδειξη και θέση, εξασφαλίζοντας ότι όλες οι πιθανές τοποθετήσεις εξετάζονται και αποθηκεύονται, έτοιμες για περαιτέρω ανάλυση.

9.6.2.7 Ενημέρωση των Πιθανών Λύσεων

Αφού ολοκληρωθούν οι απαραίτητοι έλεγχοι για το αν η ένδειξη μπορεί να τοποθετηθεί στη θέση που εξετάζεται, το επόμενο βήμα είναι η ενημέρωση της λίστας των πιθανών λύσεων για τη γραμμή ή τη στήλη. Σε αυτό το στάδιο, ο αλγόριθμος χρησιμοποιεί το αποτέλεσμα των προηγούμενων βημάτων για να προσδιορίσει πώς κάθε θέση θα επηρεαστεί από την πιθανή τοποθέτηση της ένδειξης. Η διαδικασία αυτή αποτελεί κρίσιμο μέρος του αλγορίθμου, καθώς εξασφαλίζει ότι όλες οι πιθανές λύσεις καταγράφονται με ακρίβεια.

9.6.2.7.1 Διαδικασία Ενημέρωσης

Η ενημέρωση της λίστας πιθανών λύσεων γίνεται μέσω μιας επαναληπτικής διαδικασίας που διατρέχει όλες τις θέσεις της λύσης στις οποίες μπορεί να επηρεαστεί η ένδειξη. Για κάθε θέση στη γραμμή ή τη στήλη που ανήκει στη συγκεκριμένη ένδειξη, ο αλγόριθμος προσθέτει στη λίστα τον αριθμό που αντιστοιχεί σε αυτήν την ένδειξη, χρησιμοποιώντας το μοντέλο κωδικοποίησης που περιγράφεται στην ενότητα *Κεφάλαιο 9.6.1.1 Αναπαράσταση Ενδείξεων μέσω* `clueIndex + 2`.

Ο αριθμός που προστίθεται εξαρτάται από το αποτέλεσμα του ελέγχου αν η ένδειξη μπορεί να τοποθετηθεί. Αν το αποτέλεσμα είναι `true`, στη λίστα προστίθεται το `clueIndex + 2`, το οποίο ταυτοποιεί με μοναδικό τρόπο την ένδειξη που εξετάζεται. Αν το αποτέλεσμα είναι `false`, προστίθεται το `0`, το οποίο υποδηλώνει ότι η συγκεκριμένη θέση δεν μπορεί να συμπληρωθεί από την ένδειξη.

9.6.2.7.2 Εφαρμογή του Αλγορίθμου Ενημέρωσης

Η εφαρμογή του αλγορίθμου ενημέρωσης βασίζεται στη λογική του `_canCluesFit`, το οποίο ελέγχει αν μια ένδειξη μπορεί να ξεκινήσει από το συγκεκριμένο σημείο (*index*) που εξετάζεται. Αυτό σημαίνει ότι, αν το αποτέλεσμα του ελέγχου είναι θετικό (δηλαδή `true`), τότε η ένδειξη μπορεί να τοποθετηθεί ξεκινώντας από τη συγκεκριμένη θέση και καλύπτει και τις επόμενες θέσεις, ανάλογα με το μήκος της.

Για παράδειγμα, αν εξετάζεται μια ένδειξη με μήκος 3 (`clue value = 3`) στη θέση 5 και ο έλεγχος επιβεβαιώνει ότι μπορεί να τοποθετηθεί (`true`), τότε οι θέσεις 5, 6 και 7 στη λίστα πιθανών λύσεων ενημερώνονται με το `clueIndex + 2`, π.χ., 4 αν το `clueIndex` είναι 2. Αν, από την άλλη, ο έλεγχος αποτύχει (`false`), αυτές οι θέσεις ενημερώνονται με το `0`, υποδεικνύοντας ότι δεν μπορούν να χρησιμοποιηθούν για την τρέχουσα ένδειξη.

9.6.2.7.3 Σημασία για τη Συνολική Λύση

Η διαδικασία ενημέρωσης της λίστας πιθανών λύσεων αποτελεί ένα θεμελιώδες εργαλείο για τη σωστή λειτουργία του αλγορίθμου επίλυσης παζλ εικονόσταυρων. Δεν παρέχει άμεσα την τελική λύση, αλλά θέτει τις βάσεις για μια πιο στοχευμένη και αποδοτική αναζήτηση της λύσης, διατηρώντας μια αναλυτική καταγραφή όλων των θέσεων που μπορούν να αξιοποιηθούν.

Η χρήση της αναπαράστασης `clueIndex + 2`, όπως περιγράφεται στην ενότητα *Κεφάλαιο 9.6.1.1 Αναπαράσταση Ενδείξεων μέσω* `clueIndex + 2`, καθιστά δυνατή τη μοναδική ταυτοποίηση κάθε ένδειξης που ενημερώνει τη λίστα πιθανών λύσεων. Αυτή η αναπαράσταση διασφαλίζει ότι το σύστημα μπορεί να αναγνωρίσει με σαφήνεια ποια ένδειξη μπορεί να τοποθετηθεί σε κάθε θέση, κάτι που είναι ιδιαίτερα χρήσιμο στις επόμενες φάσεις του αλγορίθμου, όπως οι επιπλέον έλεγχοι εγκυρότητας και η επεξεργασία της τελικής λύσης.

Επιπλέον, μέσω της ενημέρωσης με τη διττή προσέγγιση (τοποθέτηση του `clueIndex + 2` για έγκυρες θέσεις ή του `0` για μη έγκυρες), ο αλγόριθμος καταφέρνει να περιορίσει σημαντικά τον αριθμό των επιλογών που πρέπει να εξεταστούν στα επόμενα βήματα. Αυτό μειώνει τον αριθμό των επαναλήψεων και την πολυπλοκότητα της επεξεργασίας, διατηρώντας τη συνολική αποδοτικότητα του αλγορίθμου.

Συνολικά, η ενημέρωση της λίστας πιθανών λύσεων λειτουργεί ως συνδετικός κρίκος ανάμεσα στις επιμέρους φάσεις του αλγορίθμου, παρέχοντας τα δεδομένα που χρειάζονται για να προχωρήσει η επίλυση. Χωρίς αυτήν την ακριβή και αποτελεσματική καταγραφή, οι επόμενες φάσεις της επεξεργασίας δεν θα μπορούσαν να προχωρήσουν ομαλά. Η λειτουργία αυτή ενισχύει τη δυνατότητα του αλγορίθμου να επεξεργάζεται μεγάλα και σύνθετα παζλ με ακρίβεια και ταχύτητα, προσφέροντας ένα οργανωμένο και αξιόπιστο σύστημα διαχείρισης δεδομένων.

9.6.2.8 Τελικό Αποτέλεσμα

Το τελικό αποτέλεσμα της διαδικασίας υπολογισμού όλων των πιθανών λύσεων για μία γραμμή ή στήλη του παζλ είναι μια *πολυδιάστατη λίστα*. Αυτή η λίστα περιλαμβάνει **υπολίστες**, όπου κάθε υπολίστα αντιστοιχεί σε μια θέση της γραμμής ή της στήλης που εξετάστηκε. Κάθε στοιχείο στις υπολίστες αυτές αναπαριστά τις πιθανές ενδείξεις που μπορούν να τοποθετηθούν σε αυτή τη θέση, σύμφωνα με την αναπαράσταση που περιγράφεται στην ενότητα *Κεφάλαιο 9.6.1.1 Αναπαράσταση Ενδείξεων μέσω `clueIndex + 2`*.

9.6.2.8.1 Διαδικασία Καταγραφής Αποτελεσμάτων

Η λίστα αυτή είναι το αποτέλεσμα της ολοκληρωμένης εφαρμογής όλων των προηγούμενων βημάτων του αλγορίθμου. Κατά τη διαδικασία, κάθε ένδειξη ελέγχεται αν μπορεί να τοποθετηθεί στις θέσεις της λύσης μέσω του συνδυασμού βελτιστοποιήσεων όπως ο *Υπολογισμός Ελάχιστου και Μέγιστου Σημείου Εκκίνησης* και της χρήσης της λειτουργίας `_canCluesFit`, όπως αναλύθηκε σε προηγούμενες ενότητες.

Τα δεδομένα που συλλέγονται από αυτούς τους ελέγχους αποθηκεύονται στη λίστα πιθανών λύσεων, καταγράφοντας είτε την ένδειξη (με τη μορφή `clueIndex + 2`) είτε το `0`, το οποίο δηλώνει ότι η θέση δεν είναι διαθέσιμη για την ένδειξη.

9.6.2.8.2 Παράδειγμα Αναπαράστασης Αποτελεσμάτων

Για παράδειγμα, αν μια στήλη με στοιχεία `[2, 1, 3]` έχει μήκος `10` και περιλαμβάνει τρεις ενδείξεις, η τελική λίστα πιθανών λύσεων μπορεί να είναι της μορφής:

```
[[2], [2], [2, 0], [3], [3, 0], [3, 4], [4], [4], [4, 0], [4, 0]]
```

Σε αυτή την αναπαράσταση:

- Οι τιμές `2`, `3`, και `4` αντιστοιχούν στις ενδείξεις `clueIndex + 2` για τις τρεις ενδείξεις της γραμμής.
- Το `0` υποδηλώνει ότι η συγκεκριμένη θέση δεν μπορεί να χρησιμοποιηθεί για καμία από τις ενδείξεις, αν υπάρχει μόνο αυτό στη λίστα.

9.6.2.8.3 Σημασία για την Τελική Λύση

Η καταγραφή αυτή παρέχει την απαραίτητη πληροφορία για την επόμενη φάση της επίλυσης του παζλ, διασφαλίζοντας ότι ο αλγόριθμος έχει σαφή εικόνα των δυνατών τοποθετήσεων για κάθε ένδειξη. Επιπλέον:

- Η χρήση του `clueIndex + 2` εξασφαλίζει την άμεση συσχέτιση της θέσης με τη συγκεκριμένη ένδειξη.
- Η χρήση της *cache*, όπως περιγράφεται στην ενότητα *Κεφάλαιο 9.8.6 Διαχείριση Μνήμης Cache (`updateCachedBoxSolutions`)*, μειώνει περαιτέρω την επεξεργαστική πολυπλοκότητα, αποφεύγοντας περιττούς υπολογισμούς.

9.6.2.8.4 Ρόλος στη Συνολική Επίλυση

Το αποτέλεσμα αυτό λειτουργεί ως **κρίσιμο εργαλείο** για την επεξεργασία και την τελική επίλυση του παζλ, προσφέροντας μια οργανωμένη βάση δεδομένων για την ανάλυση και την προώθηση της λύσης στις επόμενες φάσεις του αλγορίθμου. Μέσω αυτής της δομής δεδομένων, ο αλγόριθμος αποκτά τη δυνατότητα να αντιμετωπίσει μεγάλες και σύνθετες γραμμές ή στήλες με ακρίβεια και αποδοτικότητα.

9.6.3 Υπολογισμός Ακραίων Λύσεων από Λίστα Πιθανών Λύσεων (_getSideMostSolution)

9.6.3.1 Εισαγωγή

Η μέθοδος _getSideMostSolution υπολογίζει την *"ακραία" λύση*, δηλαδή την πιο πρώιμη ή την πιο ύστερη τοποθέτηση για μια γραμμή ή στήλη ενός παζλ εικονόσταυρου. Χρησιμοποιώντας τις ενδείξεις (`clues`) και την τρέχουσα κατάσταση της λύσης (`remainingSolution`), ο αλγόριθμος προσδιορίζει πού μπορεί να ξεκινήσει ή να τελειώσει κάθε ένδειξη, δημιουργώντας μια πιθανή λύση που ικανοποιεί τους περιορισμούς του παζλ.

Η έννοια της "ακραίας" λύσης εξαρτάται από την κατεύθυνση τοποθέτησης, που καθορίζεται από την παράμετρο `axis`. Υπάρχουν δύο κατευθύνσεις:

- **`NonoAxisAlignment.start`:** Η μέθοδος δημιουργεί την πιο πρώιμη δυνατή λύση, ξεκινώντας από την αρχή της γραμμής ή στήλης.
- **`NonoAxisAlignment.end`:** Υπολογίζει την πιο ύστερη δυνατή λύση, τοποθετώντας τις ενδείξεις όσο το δυνατόν πιο κοντά στο τέλος της γραμμής ή στήλης.

Η διαδικασία περιλαμβάνει τα εξής βήματα:

1. Σταδιακή τοποθέτηση των ενδείξεων στη λύση.
2. Ενημέρωση της υπόλοιπης διαθέσιμης περιοχής της γραμμής ή στήλης.
3. Πλήρωση των κενών θέσεων με μηδενικά (`0`), ώστε να τηρούνται οι κανόνες του παζλ.

Η μέθοδος _getSideMostSolution εξυπηρετεί έναν κρίσιμο ρόλο στη διαδικασία επίλυσης, καθώς δημιουργεί μια λύση που πληροί τις απαιτήσεις του παζλ, χρησιμοποιώντας τη λίστα ενδείξεων (clues) και την τρέχουσα κατάσταση της γραμμής ή στήλης (`remainingSolution`). Η χρήση αυτής της μεθόδου ενισχύει την ακρίβεια και την αποδοτικότητα στις επόμενες φάσεις επεξεργασίας του αλγορίθμου.

9.6.3.2 Προετοιμασία δεδομένων

Η προετοιμασία των δεδομένων είναι ένα κρίσιμο στάδιο στη λειτουργία της _getSideMostSolution, καθώς εξασφαλίζει την ορθή επεξεργασία ανεξάρτητα από την κατεύθυνση ή τη διάταξη των δεδομένων. Το στάδιο αυτό περιλαμβάνει τη διαχείριση και προσαρμογή των εισόδων, ώστε ο αλγόριθμος να μπορεί να λειτουργήσει ορθά σε όλες τις περιπτώσεις.

9.6.3.2.1 Αντιστροφή Δεδομένων με Βάση την Κατεύθυνση

Όταν απαιτείται η *"πιο ύστερη λύση"* (`NonoAxisAlignment.end`), τα δεδομένα εισόδου αντιστρέφονται για να προσαρμοστούν στην επεξεργασία από το τέλος προς την αρχή. Συγκεκριμένα, οι λίστες πιθανών λύσεων (`initialSolution`), ενδείξεων (`initialClues`), και δείκτες ενδείξεων (`clueIndexes`) αντιστρέφονται μέσω της μεθόδου `.reversed.toList()`, όπως φαίνεται παρακάτω:

```
if (axis == NonoAxisAlignment.end) {
  solution = initialSolution.reversed.toList();
  clues = clues.reversed.toList();
  clueIndexes = clueIndexes.reversed.toList();
}
```

Μετά την ολοκλήρωση της επεξεργασίας, η τελική λύση επαναφέρεται στην αρχική της διάταξη μέσω της αντιστροφής της λίστας `sideMostSolution`. Αυτό διασφαλίζει ότι η επιστρεφόμενη λύση είναι συνεπής με την αρχική κατεύθυνση:

```
return axis == NonoAxisAlignment.end
    ? sideMostSolution.reversed.toList()
    : sideMostSolution;
```

Η διαδικασία αυτή διασφαλίζει ότι το αποτέλεσμα είναι συμβατό με το πλαίσιο που ορίζει ο άξονας (`NonoAxisAlignment`), είτε πρόκειται για την αρχή είτε για το τέλος της γραμμής ή της στήλης.

9.6.3.2.2 Αρχικοποίηση Λιστών Λύσης

Μετά την προσαρμογή των δεδομένων, αρχικοποιούνται δύο βασικές λίστες:

- **`sideMostSolution`:** Ξεκινά ως κενή και θα περιέχει τη λύση που δημιουργείται βήμα-βήμα.
- **`remainingSolution`:** Αντιπροσωπεύει την υπόλοιπη διαθέσιμη περιοχή της γραμμής ή στήλης που απαιτεί περαιτέρω επεξεργασία. Αρχικοποιείται από τα δεδομένα της `initialSolution`, είτε αντιστραμμένα είτε όχι.

```
final List<String> sideMostSolution = <String>[];
List<List<String>> remainingSolution = solution;
```

Αυτή η αρχικοποίηση εξασφαλίζει ότι όλες οι αλλαγές πραγματοποιούνται με ασφάλεια και ότι τα αρχικά δεδομένα παραμένουν ανεπηρέαστα.

9.6.3.2.3 Σκοπός της Προετοιμασίας

Η προετοιμασία δεδομένων έχει ως στόχο να διασφαλίσει ότι οι επόμενες λειτουργίες εκτελούνται ορθά, ανεξάρτητα από την αρχική διάταξη ή κατεύθυνση των δεδομένων. Παράλληλα, προστατεύει τα υπόλοιπα δεδομένα του παζλ από ανεπιθύμητες αλλαγές, καθώς όλες οι μεταβολές περιορίζονται στο τοπικό πλαίσιο της μεθόδου. Με αυτόν τον τρόπο, η προετοιμασία δεδομένων λειτουργεί ως θεμέλιο για την επιτυχημένη υλοποίηση της _getSideMostSolution.

9.6.3.3 Διαδικασία Υπολογισμού

Η **διαδικασία υπολογισμού** περιλαμβάνει την επαναληπτική επεξεργασία κάθε ένδειξης στη λίστα `clues`, αξιοποιώντας τόσο τη θέση όσο και την τιμή τους για την ενημέρωση της λύσης.

9.6.3.3.1 Εντοπισμός Θέσης της Ένδειξης

Για κάθε ένδειξη, η μέθοδος `indexWhere` εντοπίζει τη θέση της ένδειξης μέσα στη λίστα `remainingSolution`, αναζητώντας την κωδικοποιημένη τιμή της (`clueIndex + 2`). Αυτή η διαδικασία διασφαλίζει την ακριβή τοποθέτηση της ένδειξης, σύμφωνα με την αντιστοιχία που περιγράφεται στην ενότητα *Κεφάλαιο 9.6.1.1 Αναπαράσταση Ενδείξεων μέσω `clueIndex + 2`*.

```
final int cluePos = remainingSolution.indexWhere(
  (List<String> list) => list.contains('$clueIndex')
);
```

9.6.3.3.2 Αρχική Ενημέρωση Λίστας Λύσης

Αφού εντοπιστεί η θέση, προστίθενται μηδενικά (0) στη λίστα sideMostSolution για να καλυφθούν τα κενά πριν από την ένδειξη. Στη συνέχεια, η ένδειξη προστίθεται για το διάστημα που αντιστοιχεί στην τιμή της (clueValue). Αν η λύση δεν έχει ολοκληρωθεί, προστίθεται ένα ακόμη μηδενικό (0) για να διαχωριστεί η ένδειξη από τις επόμενες.

```
if (cluePos > 0) {
  sideMostSolution.addAll(
    Iterable<String>.generate(cluePos, (_) => '0').toList()
  );
}
sideMostSolution.addAll(
  Iterable<String>.generate(clue, (_) => '$clueIndex').toList()
);
if (sideMostSolution.length < solution.length) {
  sideMostSolution.add('0');
}
```

9.6.3.3.3 Ενημέρωση Λίστας με τα Δεδομένα του Στοιχείου

Μετά την προσθήκη όλων των μηδενικών (0) μέχρι τη θέση cluePos, ακολουθεί η τοποθέτηση του ίδιου του στοιχείου στη λίστα λύσης (sideMostSolution). Αυτή η διαδικασία ξεκινά από τη θέση cluePos και συνεχίζει για όσες θέσεις ορίζει το clueValue, δηλαδή το μήκος της ένδειξης. Σε κάθε θέση που συμπληρώνεται στη λίστα, εισάγεται η τιμή clueIndex + 2, σύμφωνα με τη θεωρία που περιγράφεται στην ενότητα *Κεφάλαιο 9.6.1.1 Αναπαράσταση Ενδείξεων μέσω `clueIndex + 2`.*

Για παράδειγμα, αν η remainingSolution έχει την τιμή:

```
[[2], [2], [2, 0], [3], [3, 0], [3, 4], [4], [4], [4, 0], [4, 0]]
```

και τα στοιχεία (clues) είναι [2, 1, 3], το πρώτο clue έχει clueIndex = 0 και clueValue = 2. Σύμφωνα με τη θεωρία του clueIndex + 2, η τιμή του clue που θα τοποθετηθεί στη λύση είναι 2. Ο αλγόριθμος βρίσκει τη θέση της τιμής 2 στη remainingSolution και την αποθηκεύει στη μεταβλητή cluePos. Στην περίπτωση αυτή, η τιμή του cluePos είναι 0, επειδή το 2 υπάρχει στην πρώτη υπολίστα της remainingSolution. Από τη θέση cluePos, ο αλγόριθμος προσθέτει 2 στοιχεία με την τιμή 2 στη λίστα sideMostSolution, που γίνεται:

```
[‘2’, ‘2’]
```

9.6.3.3.4 Ενημέρωση Υπόλοιπης Λίστας

Συνεχίζοντας το παραπάνω παράδειγμα, ο αλγόριθμος ελέγχει αν παραμένουν κενές θέσεις στη sideMostSolution. Αν υπάρχουν, προστίθεται ένα μηδενικό (0) για να εξασφαλιστεί ότι υπάρχει το υποχρεωτικό κενό ανάμεσα στο

τρέχον στοιχείο και στο επόμενο. Στη συγκεκριμένη περίπτωση, το μήκος της `sideMostSolution` είναι 2, ενώ το μήκος της συνολικής λύσης (`initialSolution`) είναι 10. Επομένως, προστίθεται ένα 0 στο τέλος της λύσης, και η λίστα γίνεται:

```
['2', '2', '0']
```

Αυτή η διαδικασία διασφαλίζει ότι κάθε στοιχείο διαχωρίζεται κατάλληλα από το επόμενο, ακολουθώντας τους κανόνες του παζλ και επιτρέποντας την εύκολη αναγνώριση και διαχείριση των ενδείξεων στην επόμενη φάση.

9.6.3.4 Συμπεράσματα

Η μέθοδος `getSideMostSolution` παρέχει έναν αποτελεσματικό τρόπο υπολογισμού της πιο ακραίας λύσης για μια δεδομένη γραμμή ή στήλη, λαμβάνοντας υπόψη τις ενδείξεις και την υπάρχουσα κατάσταση της λύσης. Μέσω της αντιστροφής των δεδομένων για την κατεύθυνση `NonoAxisAlignment.end` και της σταδιακής ενημέρωσης της λίστας λύσης, εξασφαλίζει ότι οι κανόνες του παζλ τηρούνται με ακρίβεια.

Η ευελιξία της μεθόδου να προσαρμόζει τα δεδομένα εισόδου και να ενημερώνει τη λύση βήμα-βήμα την καθιστά ένα σημαντικό εργαλείο για την επίλυση εικονόσταυρων. Παράλληλα, η προσθήκη μηδενικών (`0`) στα σωστά σημεία διασφαλίζει τη διατήρηση των κενών ανάμεσα στα στοιχεία, δημιουργώντας μια συνεπή και πλήρη λύση. Η μέθοδος αυτή δεν αποτελεί μόνο ένα εργαλείο για τον υπολογισμό λύσεων, αλλά και έναν σημαντικό παράγοντα στη μείωση της πολυπλοκότητας των επόμενων φάσεων επίλυσης.

9.6.4 Εύρεση Κοινών Αποτελεσμάτων Ακραίων Λύσεων (`_getSideMostSolutionsMatches`)

9.6.4.1 Εισαγωγή

Η μέθοδος `_getSideMostSolutionsMatches` αποτελεί έναν κρίσιμο μηχανισμό για την αναγνώριση των *σίγουρων* θέσεων στις οποίες ένα στοιχείο (ή κενό) μπορεί να τοποθετηθεί ή να αποκλειστεί με βεβαιότητα σε μια γραμμή ή στήλη ενός παζλ εικονόσταυρου. Με βάση τις πιο πρώιμες (`startingMostSolution`) και πιο ύστερες (`endingMostSolution`) δυνατές λύσεις, η μέθοδος εντοπίζει τις κοινές θέσεις όπου τα στοιχεία παραμένουν σταθερά και στις δύο αυτές ακραίες λύσεις.

Επιπλέον, η μέθοδος χρησιμοποιεί την πλήρη λίστα πιθανών λύσεων (`allLineSolutions`) και τις θέσεις των ερωτηματικών (`charIndexesOfQMarks`) για να συμπληρώσει τα δεδομένα σχετικά με τα *σίγουρα* κουτάκια. Το αποτέλεσμα είναι ένας χάρτης (`map`), όπου:

- Τα κλειδιά (`keys`) αναπαριστούν τους αριθμούς των στοιχείων (clue values).
- Οι τιμές (`values`) είναι λίστες με τις θέσεις που έχουν επιβεβαιωθεί ως σταθερές.

9.6.4.2 Προετοιμασία Δεδομένων

Για να εντοπιστούν οι κοινές θέσεις, η μέθοδος αρχικά δημιουργεί έναν χάρτη (`matchMap`) για την αποθήκευση των αποτελεσμάτων. Κάθε στοιχείο στον χάρτη έχει:

- Κλειδί (`key`): Την αριθμητική τιμή του στοιχείου (με βάση την κωδικοποίηση `clueIndex + 2`).
- Τιμή (`value`): Το σύνολο των θέσεων που το στοιχείο εμφανίζεται με σιγουριά.

Τα δεδομένα εισόδου, δηλαδή οι λύσεις `startingMostSolution` και `endingMostSolution`, μετατρέπονται σε σύνολα (`Set`) χρησιμοποιώντας τη μέθοδο `indexed`. Αυτή η μετατροπή επιτρέπει:

- Τη σύγκριση των θέσεων που περιέχουν τα ίδια στοιχεία στις δύο ακραίες λύσεις.
- Την αποτελεσματική ταυτοποίηση των κοινών θέσεων.

Με αυτήν τη δομή, η μέθοδος διασφαλίζει την ακριβή και αποδοτική καταγραφή των στοιχείων που είναι σταθερά στις δύο ακραίες λύσεις, παρέχοντας μια σαφή εικόνα για τις θέσεις που μπορούν να θεωρηθούν ως *σίγουρες*.

9.6.4.3 Εύρεση Κοινών Θέσεων

Η ενότητα αυτή περιγράφει τον εντοπισμό κοινών θέσεων μεταξύ της αρχικής και της τελικής ακραίας λύσης, δηλαδή της `startingMostSolution` και της `endingMostSolution`. Αυτή η διαδικασία είναι θεμελιώδης για την αναγνώριση των *σίγουρων* θέσεων, που παραμένουν σταθερές ανεξαρτήτως λύσης.

9.6.4.3.1 Δημιουργία Συνόλων από Αρχική και Τελική Λύση

Η διαδικασία ξεκινά με τη δημιουργία δύο συνόλων από τις λύσεις `startingMostSolution` και `endingMostSolution`. Αυτά τα σύνολα περιέχουν ζεύγη που αντιστοιχούν στη θέση και την τιμή κάθε στοιχείου στις λύσεις, χρησιμοποιώντας τη μέθοδο `indexed`. Η `indexed` δημιουργεί ένα σύνολο ζευγών (*θέση, τιμή*), που παρέχει σαφή αναπαράσταση των δεδομένων. Ο κώδικας για αυτήν τη διαδικασία είναι:

```
final Set<(int, String)> inputNumbersStart = startingMostSolution.indexed.toSet();
final Set<(int, String)> inputNumbersEnd = endingMostSolution.indexed.toSet();
```

Με αυτόν τον τρόπο, οι πληροφορίες για κάθε θέση και τιμή στις δύο λύσεις είναι δομημένες και έτοιμες για σύγκριση.

9.6.4.3.2 Υπολογισμός Κοινών Θέσεων με Μέθοδο Τομής

Για να εντοπιστούν οι κοινές θέσεις, εφαρμόζεται η μέθοδος `intersection` στα δύο σύνολα. Η μέθοδος αυτή εντοπίζει τις θέσεις και τις τιμές που είναι ίδιες και στις δύο λύσεις. Το αποτέλεσμα είναι ένα νέο σύνολο, `duplicateInputNumbers`, που περιέχει μόνο τα κοινά στοιχεία. Ο σχετικός κώδικας είναι:

```
final Set<(int, String)> duplicateInputNumbers = inputNumbersStart.intersection(inputNumbersEnd);
```

Για παράδειγμα, αν η `startingMostSolution` είναι `[2, 2, 0, 3, 0, 4, 4, 4, 0, 0]` και η `endingMostSolution` είναι `[0, 0, 2, 2, 0, 3, 0, 4, 4, 4]`, τότε το `duplicateInputNumbers` περιέχει τη θέση `(7, 4)`.

9.6.4.3.3 Φιλτράρισμα και Ενημέρωση του πίνακα κοινών στοιχείων (`matchMap`)

Αφού εντοπιστούν οι κοινές θέσεις, αυτές φιλτράρονται βάσει δύο κριτηρίων:

1. Οι θέσεις που έχουν τιμή `0` αγνοούνται, καθώς αντιπροσωπεύουν κενά κουτάκια.
2. Εξετάζεται αν οι θέσεις περιλαμβάνονται στη λίστα `charIndexesOfQMarks`, η οποία περιέχει τις αρχικές θέσεις που είχαν το σύμβολο `?` και επομένως δεν είχαν σαφή απάντηση.

Οι θέσεις που πληρούν και τα δύο αυτά κριτήρια προστίθενται στο `matchMap`. Για παράδειγμα, αν το `duplicateInputNumbers` περιέχει το ζεύγος `(7, 4)` και η θέση 7 βρίσκεται στο `charIndexesOfQMarks`, τότε αυτή η θέση προστίθεται στο `matchMap` με κλειδί την τιμή 4, και το αποτέλεσμα γίνεται `{4: [7]}`.

Πιο αναλυτικά, ο αλγόριθμος διατρέχει κάθε στοιχείο του `duplicateInputNumbers`. Κάθε στοιχείο είναι ένα ζεύγος `(θέση, τιμή)` που αντιστοιχεί στη θέση και την τιμή του στοιχείου στις λύσεις `startingMostSolution` και `endingMostSolution`. Ο κώδικας για αυτή τη φάση ξεκινά με:

```
for (final (int leftNumber, String rightNumberStr) in duplicateInputNumbers) {
```

Εδώ, το `leftNumber` αντιπροσωπεύει τη θέση του στοιχείου, ενώ το `rightNumberStr` είναι η τιμή του στοιχείου σε μορφή συμβολοσειράς.

Η τιμή `rightNumberStr` μετατρέπεται σε ακέραιο για ευκολία στη διαχείριση:

```
final int rightNumber = int.parse(rightNumberStr);
```

Ο πρώτος έλεγχος διασφαλίζει ότι η τιμή δεν είναι `0`:

```
if (rightNumber != 0 && charIndexesOfQMarks.contains(leftNumber)) {
```

Όταν πληρούνται οι συνθήκες, η θέση `leftNumber` προστίθεται στο `matchMap` κάτω από το κλειδί `rightNumber`:

```
matchMap.putIfAbsent(rightNumber, () => <int>{});
matchMap[rightNumber]!.add(leftNumber);
```

Για παράδειγμα, αν το `duplicateInputNumbers` περιέχει το στοιχείο `(7, '4')`, η θέση 7 προστίθεται στον χάρτη ως εξής:

```
matchMap[4] = {7};
```

Αυτό σημαίνει ότι ο χάρτης `matchMap` ενημερώνεται ώστε να αποθηκεύσει τη βεβαιότητα ότι η ένδειξη 4 (η οποία, σύμφωνα με την κωδικοποίηση που περιγράφεται στην ενότητα Κεφάλαιο 9.6.1.1 Αναπαράσταση Ενδείξεων μέσω `clueIndex + 2`, αντιστοιχεί στην ένδειξη με `clueIndex = 2` στη λίστα των ενδείξεων) καταλαμβάνει τη θέση 7 στη λύση.

Με απλά λόγια, η τιμή 4 προσδιορίζει την ένδειξη που εξετάζεται, ενώ το σύνολο `{7}` αποθηκεύει τη θέση όπου αυτή η ένδειξη είναι βέβαιο ότι πρέπει να τοποθετηθεί. Αν το σύνολο περιείχε περισσότερους αριθμούς, π.χ. `{7, 8, 12}`, αυτό θα σήμαινε ότι η ένδειξη 4 είναι βέβαιο ότι πρέπει να τοποθετηθεί όχι μόνο στη θέση 7, αλλά και στις θέσεις 8 και 12.

Αυτή η πληροφορία είναι ζωτικής σημασίας για τη σωστή παρακολούθηση και αποτύπωση των θέσεων της λύσης όπου η ένδειξη μπορεί να τοποθετηθεί με σιγουριά, εξασφαλίζοντας ότι οι κατάλληλες θέσεις στη λύση ενημερώνονται αναλόγως με την ένδειξη 4.

Αυτή η διαδικασία εξασφαλίζει την ακρίβεια στην καταγραφή των *σίγουρων θέσεων* για κάθε ένδειξη, δημιουργώντας ένα αποδοτικό και καλά οργανωμένο σύστημα αναφοράς.

9.6.4.4 Εντοπισμός Κελιών με Μηδενικά

Στο στάδιο αυτό, ο αλγόριθμος αναγνωρίζει τα κελιά της γραμμής ή στήλης που είναι πάντα κενά, δηλαδή περιέχουν μηδενικά (0) ανεξαρτήτως της λύσης. Αυτή η διαδικασία εξασφαλίζει ότι οι θέσεις που δεν μπορούν ποτέ να γεμίσουν καταγράφονται και σηματοδοτούνται με ακρίβεια στο αποτέλεσμα.

Στην πράξη, η εφαρμογή θέλει να βρει όλες τις θέσεις οι οποίες σε όλες τις λύσεις, η απάντηση στο τι στοιχείο θα μπει σε εκείνη τη θέση, είναι μηδέν (0). Σε όλες αυτές τις θέσεις, μπορεί να τοποθετηθεί το μηδέν (0).

9.6.4.4.1 Χρήση Κανονικής Έκφρασης (Regular Expression)

Η ανίχνευση των κενών κελιών γίνεται μέσω της ανάλυσης της λίστας `allLineSolutions`, η οποία περιέχει όλες τις πιθανές λύσεις για κάθε θέση στη γραμμή ή στήλη. Ο αλγόριθμος χρησιμοποιεί κανονική έκφραση (Regular Expression) για να βρει τις θέσεις όπου το μοναδικό στοιχείο είναι το μηδενικό.

Η κανονική έκφραση που χρησιμοποιείται είναι:

```
r'\((' + inclusionPattern + r'), \[(0)\]\)'
```

Επεξήγηση της Κανονικής Έκφρασης:

1. `\(` και `\)`: Αναζητά ανοικτή και κλειστή παρένθεση, που περιβάλλει κάθε ζεύγος στη λίστα `allLineSolutions` (π.χ., (index, [value])).
2. `inclusionPattern`: Αντιπροσωπεύει τις θέσεις που περιέχουν ερωτηματικά (?) στη λύση. Είναι μια λίστα αριθμών που προκύπτει από τις θέσεις `charIndexesOfQMarks`. Αυτοί οι αριθμοί συνδέονται με τον χαρακτήρα `|` (OR) για να δημιουργηθεί το μοτίβο των επιτρεπόμενων θέσεων.
   Π.χ., αν `charIndexesOfQMarks = [2, 4, 6]`, το `inclusionPattern` θα είναι `2|4|6`.
3. `, \[(0)\]`: Αναζητά θέσεις όπου η τιμή στη λίστα είναι `[0]`. Το μηδενικό (0) περικλείεται σε αγκύλες, κάτι που αντιστοιχεί στον τρόπο που αποθηκεύονται οι πιθανές λύσεις στην `allLineSolutions`.

Για παράδειγμα, αν η `allLineSolutions` έχει την εξής μορφή:

```
[(2, [0]), (4, [1, 0]), (6, [0])]
```

και οι θέσεις `charIndexesOfQMarks` είναι `[2, 4, 6]`, τότε η κανονική έκφραση θα αντιστοιχεί σε:

```
r'\((2|4|6), \[(0)\]\)'
```

Η έκφραση αυτή θα εντοπίσει τις θέσεις `(2, [0])` και `(6, [0])`, αγνοώντας τη θέση `(4, [1, 0])`, επειδή περιέχει και άλλες τιμές εκτός του μηδενικού.

9.6.4.4.2 Ανάλυση και Προσθήκη στον Πίνακα Αποτελεσμάτων (`matchMap`)

Οι θέσεις που ταιριάζουν με την κανονική έκφραση αποθηκεύονται ως σύνολο (`Set`) στο `matchMap`. Για να εξασφαλιστεί η ακεραιότητα των δεδομένων:

- Εάν το `matchMap` δεν περιέχει ήδη το κλειδί 0, τότε αυτό δημιουργείται.

- Οι θέσεις προστίθενται ως τιμές στο κλειδί 0, το οποίο αναπαριστά τα σίγουρα κενά κελιά.

Ένα παράδειγμα ενημέρωσης είναι πως, για την `allLineSolutions`:

```
[(2, [0]), (4, [1, 0]), (6, [0])]
```

Η ενημέρωση του `matchMap` θα αποδώσει:

```
{0: [2, 6]}
```

Αυτό σημαίνει ότι οι θέσεις 2 και 6 είναι πάντα κενές.

9.6.4.5 Επιστροφή Αποτελέσματος

Η μέθοδος `_getSideMostSolutionsMatches` ολοκληρώνεται με τη διαδικασία επιστροφής ενός χάρτη (`Map`) που περιέχει τις βέβαιες θέσεις για κάθε ένδειξη της γραμμής ή στήλης. Ο χάρτης αυτός παρέχει τη σύνδεση ανάμεσα στις ενδείξεις και τις θέσεις στις οποίες αυτές μπορούν να τοποθετηθούν με απόλυτη σιγουριά, βασισμένο στα δεδομένα που συλλέχθηκαν και επεξεργάστηκαν στα προηγούμενα βήματα.

9.6.4.5.1 Μετατροπή του Χάρτη Αποτελεσμάτων (`matchMap`)

Η επιστροφή του αποτελέσματος ξεκινά με τη μετατροπή του `matchMap`, ο οποίος χρησιμοποιεί ως τιμές σύνολα (`Set<int>`), σε έναν πιο εύχρηστο χάρτη όπου οι τιμές είναι λίστες (`List<int>`). Αυτή η μετατροπή γίνεται για να είναι πιο εύκολη η διαχείριση και η πρόσβαση στα δεδομένα στις επόμενες φάσεις του αλγορίθμου.

Ο κώδικας που πραγματοποιεί αυτή τη μετατροπή είναι ο εξής:

```
return matchMap.map((int key, Set<int> value) =>
    MapEntry<int, List<int>>(key, value.toList()));
```

Ας αναλύσουμε τι ακριβώς συμβαίνει εδώ:

- `map Method`: Η `map` εφαρμόζεται στον `matchMap` και δημιουργεί έναν νέο χάρτη, μετατρέποντας κάθε είσοδο του παλιού χάρτη σε μία νέα είσοδο του νέου.
- `MapEntry`: Η `MapEntry` χρησιμοποιείται για τη δημιουργία μιας νέας εγγραφής στον χάρτη. Τα κλειδιά παραμένουν ως έχουν (`int key`), αλλά οι τιμές που ήταν προηγουμένως σύνολα (`Set<int>`) μετατρέπονται σε λίστες (`List<int>`).
- `toList()`: Κάθε τιμή-σύνολο (`Set<int> value`) μετατρέπεται σε λίστα μέσω της μεθόδου `toList()`, ώστε να είναι πιο εύχρηστη για μελλοντική επεξεργασία.

9.6.4.5.2 Τελική Δομή του Χάρτη

Το αποτέλεσμα που επιστρέφεται είναι ένας χάρτης με τους εξής χαρακτηρισμούς:

- **Κλειδιά (keys)**: Τα `clueIndex + 2` που αντιπροσωπεύουν τις ενδείξεις, σύμφωνα με την κωδικοποίηση που περιγράφεται στο Κεφάλαιο 9.6.1.1 Αναπαράσταση Ενδείξεων μέσω clueIndex + 2.

- **Τιμές (values)**: Λίστες που περιέχουν τις θέσεις των κουτιών στις οποίες η συγκεκριμένη ένδειξη είναι σίγουρο ότι μπορεί να τοποθετηθεί.

Για παράδειγμα, αν ο `matchMap` περιείχε:

```
{4: {7, 8}, 5: {10}, 0: {3, 15}}
```

Μετά τη μετατροπή, ο τελικός χάρτης που επιστρέφεται θα ήταν:

```
{4: [7, 8], 5: [10], 0: [3, 15]}
```

9.6.4.5.3 Λειτουργική Σημασία

Η διαδικασία αυτή εξασφαλίζει ότι τα αποτελέσματα επιστρέφονται σε μορφή που είναι εύκολα κατανοητή και διαχειρίσιμη από τις επόμενες μεθόδους του αλγορίθμου επίλυσης. Με αυτόν τον τρόπο, οι ενδείξεις και οι θέσεις τους καταγράφονται με ακρίβεια και μπορούν να χρησιμοποιηθούν για την περαιτέρω βελτίωση ή την ολοκλήρωση της λύσης του παζλ.

9.6.4.5.4 Συμπεράσματα

Η μέθοδος `_getSideMostSolutionsMatches` αποτελεί ένα κρίσιμο εργαλείο για την ανάλυση των δεδομένων μιας γραμμής ή στήλης σε ένα παζλ εικονόσταυρου, καθώς εντοπίζει τα σίγουρα στοιχεία που μπορούν να συμπληρωθούν ή να αποκλειστούν. Μέσω της σύγκρισης των ακραίων λύσεων (`starting` και `ending most solutions`) και της διερεύνησης όλων των πιθανών λύσεων για κάθε θέση, ο αλγόριθμος καταλήγει σε ένα σαφές αποτέλεσμα που συνδυάζει ακρίβεια και αποδοτικότητα.

Συγκεκριμένα, η μέθοδος χρησιμοποιεί τα δεδομένα των ακραίων λύσεων για να βρει τις θέσεις στις οποίες τα στοιχεία παραμένουν σταθερά, ανεξάρτητα από το πώς θα προχωρήσει η επίλυση. Αυτές οι πληροφορίες, καταχωρημένες στον χάρτη `matchMap`, παρέχουν κρίσιμες ενδείξεις για το πού πρέπει να συμπληρωθούν στοιχεία (τιμές `clueIndex + 2`) και πού πρέπει να παραμείνουν κενά (τιμή `0`).

Η σημασία της μεθόδου έγκειται στη δυνατότητά της να:

- **Απλοποιεί την επίλυση:** Με την καταγραφή των βέβαιων θέσεων, μειώνεται η ανάγκη για περαιτέρω πολύπλοκους ελέγχους στις θέσεις αυτές.
- **Επιταχύνει τον αλγόριθμο:** Η χρήση της `cache` και της σύγκρισης ακραίων λύσεων περιορίζει τις περιττές επαναλήψεις.
- **Διατηρεί την ακρίβεια:** Η χρήση της κωδικοποίησης `clueIndex + 2` εξασφαλίζει ότι τα δεδομένα είναι σαφή και μοναδικά για κάθε ένδειξη.

Ο τελικός χάρτης που επιστρέφεται από τη μέθοδο είναι έτοιμος να χρησιμοποιηθεί στα επόμενα βήματα του αλγορίθμου, είτε για την ενημέρωση της λύσης είτε για την περαιτέρω επεξεργασία άλλων γραμμών ή στηλών. Με αυτόν τον τρόπο, η μέθοδος `_getSideMostSolutionsMatches` συμβάλλει στην αποτελεσματική και ακριβή επίλυση του παζλ, καθιστώντας την έναν αναπόσπαστο κρίκο της συνολικής διαδικασίας.

9.7 Μέθοδος Ελέγχου Τοποθέτησης Στοιχείου σε Θέση

9.7.1 Εισαγωγή

Η μέθοδος `_canCluesFit` αποτελεί τον πυρήνα της διαδικασίας ελέγχου τοποθέτησης μιας ένδειξης (clue) σε συγκεκριμένη θέση μιας γραμμής ή στήλης. Η μέθοδος αυτή καθορίζει αν μια ένδειξη μπορεί να τοποθετηθεί σύμφωνα με τους κανόνες του παζλ, λαμβάνοντας υπόψη:

- Την τρέχουσα κατάσταση της λύσης.
- Τη θέση της ένδειξης στη γραμμή ή τη στήλη.
- Τις υπόλοιπες ενδείξεις που σχετίζονται με τη γραμμή ή στήλη.

Η διαδικασία αυτή είναι θεμελιώδης, καθώς κάθε τοποθέτηση επηρεάζει τη συνολική λύση του παζλ, απαιτώντας ακρίβεια και συμμόρφωση με τους κανόνες.

9.7.1.1 Έλεγχος μήκους

Ο πρώτος έλεγχος που πραγματοποιείται είναι αν η ένδειξη μπορεί να τοποθετηθεί στη συγκεκριμένη θέση, λαμβάνοντας υπόψη το μήκος της. Για παράδειγμα, αν η ένδειξη έχει μήκος n και η τρέχουσα θέση της στη λύση βρίσκεται κοντά στο τέλος της γραμμής, μπορεί να μην υπάρχει αρκετός διαθέσιμος χώρος. Ο έλεγχος μήκους εγγυάται ότι η τοποθέτηση δεν θα παραβιάσει τα όρια της γραμμής ή της στήλης.

9.7.1.2 Συμβατότητα κουτιών

Στη συνέχεια, η μέθοδος εξετάζει αν τα υπάρχοντα δεδομένα της λύσης (π.χ., μηδενικά ή ήδη τοποθετημένα στοιχεία) είναι συμβατά με την ένδειξη που εξετάζεται. Εάν τα υπάρχοντα στοιχεία παραβιάζουν τους κανόνες τοποθέτησης, η μέθοδος απορρίπτει τη συγκεκριμένη θέση ως μη έγκυρη.

9.7.1.3 Ανάλυση άλλων ενδείξεων

Η μέθοδος `_doOtherCluesFit` καλείται για να ελέγξει αν οι υπόλοιπες ενδείξεις της γραμμής ή στήλης είναι συμβατές με την τοποθέτηση της τρέχουσας ένδειξης. Αυτό περιλαμβάνει:

- Την εξέταση των ενδείξεων που προηγούνται της τρέχουσας.
- Την ανάλυση των ενδείξεων που έπονται της τρέχουσας.

Η μέθοδος αυτή εξασφαλίζει ότι η τοποθέτηση της ένδειξης δεν θα δημιουργήσει προβλήματα στις επόμενες ενδείξεις, διατηρώντας τη συνοχή της λύσης.

9.7.1.4 Αποτελέσματα και Χρησιμότητα

Η συνεργασία των μεθόδων `_canCluesFit` και `_doOtherCluesFit` προσφέρει:

- **Αναλυτική ακρίβεια:** Κάθε ένδειξη εξετάζεται λεπτομερώς στη συγκεκριμένη θέση της.
- **Ολιστική προσέγγιση:** Εξασφαλίζεται η συνολική συνέπεια της γραμμής ή στήλης.

- **Αυξημένη απόδοση:** Αποφυγή περιττών επαναλήψεων με χρήση της μνήμης cache.

Με τη χρήση αυτών των μεθόδων, ο αλγόριθμος διασφαλίζει την ακριβή και αποδοτική επίλυση του παζλ, παρέχοντας μια ισορροπία ανάμεσα στην ακρίβεια και την απόδοση. Οι τοποθετήσεις των ενδείξεων γίνονται πιο στοχευμένες, και η διαδικασία επίλυσης γίνεται σταδιακά πιο αποτελεσματική.

9.7.2 Έλεγχος Τοποθέτησης Στοιχείου (`_canCluesFit`)

9.7.2.1 Εισαγωγή

Η μέθοδος `_canCluesFit` αποτελεί μια κρίσιμη συνιστώσα του αλγορίθμου επίλυσης παζλ εικονόσταυρου. Ο στόχος της είναι να ελέγξει αν μια συγκεκριμένη ένδειξη μπορεί να τοποθετηθεί σε μια καθορισμένη θέση της λύσης μιας γραμμής ή στήλης. Χρησιμοποιώντας δεδομένα όπως:

- Η λίστα ενδείξεων (`clues`),
- Η τρέχουσα κατάσταση της λύσης (`solution`),
- Οι παράμετροι θέσης (`solutionPosition`) και ρυθμίσεων (`settings`),

η μέθοδος αναλύει αν η ένδειξη μπορεί να ενταχθεί χωρίς να παραβιάσει τους κανόνες του παζλ.

Η μέθοδος εφαρμόζει τρεις κύριους ελέγχους:

1. Ελέγχει αν η ένδειξη χωράει στο υπόλοιπο μήκος της γραμμής.
2. Επαληθεύει ότι η ένδειξη δεν έρχεται σε σύγκρουση με υπάρχοντα γεμισμένα ή κενά κουτιά.
3. Εξετάζει αν οι υπόλοιπες ενδείξεις πριν και μετά την τρέχουσα ένδειξη ταιριάζουν σωστά, χρησιμοποιώντας τη μέθοδο `_doOtherCluesFit`.

9.7.2.2 Προετοιμασία Δεδομένων

Η μέθοδος `_canCluesFit` ξεκινά με την προετοιμασία των δεδομένων που απαιτούνται για να ελεγχθεί αν μια ένδειξη μπορεί να τοποθετηθεί στη γραμμή ή τη στήλη του παζλ εικονόσταυρου.

Στο πρώτο βήμα, η συμβολοσειρά `solution`, που αντιπροσωπεύει την τρέχουσα κατάσταση της λύσης, μετατρέπεται σε λίστα χαρακτήρων χρησιμοποιώντας τη μέθοδο `split('')`. Αυτή η μετατροπή επιτρέπει την εύκολη πρόσβαση στις μεμονωμένες θέσεις της λύσης, παρέχοντας τη δυνατότητα λεπτομερούς ανάλυσης κάθε θέσης. Για παράδειγμα:

```
final List<String> solutionList = solution.split('');
```

Αν η `solution` είναι `"0010?"`, τότε η μετατροπή αυτή δημιουργεί τη λίστα:

```
['0', '0', '1', '0', '?']
```

Αμέσως μετά, η μέθοδος ανακτά την ένδειξη που βρίσκεται στη θέση `cluePosition` από τη λίστα ενδείξεων `clues`. Αυτή η ένδειξη, που αποθηκεύεται στη μεταβλητή `clue`, αντιπροσωπεύει το μήκος της ένδειξης που πρόκειται να εξεταστεί. Για παράδειγμα:

```
final int clue = clues[cluePosition];
```

Αν η λίστα `clues` είναι `[3, 2, 1]` και η τιμή του `cluePosition` είναι 1, τότε η `clue` θα έχει τιμή 2.

Η διαδικασία προετοιμασίας αυτών των δεδομένων είναι απαραίτητη για τη σωστή λειτουργία της μεθόδου. Συγκεκριμένα:

- Η λίστα χαρακτήρων (`solutionList`) επιτρέπει την ακριβή πρόσβαση και επεξεργασία των θέσεων της λύσης.
- Η ανάκτηση της ένδειξης (`clue`) παρέχει την απαραίτητη πληροφορία για να υπολογιστεί αν μπορεί να τοποθετηθεί.

Οποιοδήποτε λάθος σε αυτό το στάδιο θα μπορούσε να οδηγήσει σε εσφαλμένα αποτελέσματα στους ελέγχους που ακολουθούν, καθιστώντας τη σωστή προετοιμασία δεδομένων κρίσιμο μέρος του αλγορίθμου.

9.7.2.3 Έλεγχος Μήκους Ενδείξεων

Ένα από τα πρώτα και πιο σημαντικά βήματα στη μέθοδο `_canCluesFit` είναι ο έλεγχος αν η ένδειξη μπορεί να τοποθετηθεί στη διαθέσιμη περιοχή της λύσης, ξεκινώντας από τη συγκεκριμένη θέση που εξετάζεται. Η διαδικασία αυτή διασφαλίζει ότι το μήκος της ένδειξης, όπως καθορίζεται από την τιμή της μεταβλητής `clue`, είναι κατάλληλο για το υπόλοιπο μήκος της γραμμής ή της στήλης, από τη θέση `solutionPosition` και μετά. Αυτός ο έλεγχος είναι ζωτικής σημασίας, καθώς αποτρέπει περιττούς υπολογισμούς και ελέγχους σε περιπτώσεις όπου η ένδειξη είναι εκ των πραγμάτων αδύνατο να χωρέσει.

Η μέθοδος υπολογίζει το διαθέσιμο μήκος από τη θέση `solutionPosition` μέχρι το τέλος της λύσης, χρησιμοποιώντας τη μέθοδο `getRange`. Συγκεκριμένα, το διαθέσιμο μήκος προσδιορίζεται από το εύρος:

```
[solutionPosition, solutionList.length]
```

Εφόσον το μήκος της ένδειξης `clue` είναι μεγαλύτερο από το διαθέσιμο μήκος, ο αλγόριθμος καταλήγει αμέσως ότι η ένδειξη δεν μπορεί να τοποθετηθεί και επιστρέφει `false`.

Για παράδειγμα, αν η λύση είναι:

```
['0', '0', '1', '0', '?']
```

και η `solutionPosition` είναι 3, τότε το διαθέσιμο μήκος είναι 2. Αν η τιμή του `clue` είναι 3, είναι προφανές ότι δεν υπάρχει αρκετός χώρος για την τοποθέτησή του.

Η λογική του ελέγχου φαίνεται στον παρακάτω κώδικα:

```
if (clue > solutionList.getRange(solutionPosition, solutionList.length).length) {
  if (printLogs) log('false');
  return false;
}
```

Ο έλεγχος μήκους είναι κρίσιμος για τη βελτιστοποίηση του αλγορίθμου. Αποτρέπει άσκοπους υπολογισμούς σε περιπτώσεις όπου είναι ήδη προφανές ότι η ένδειξη δεν μπορεί να χωρέσει στη γραμμή ή τη στήλη. Με αυτόν τον

τρόπο, ο αλγόριθμος επικεντρώνεται μόνο σε θέσεις όπου υπάρχει πραγματική πιθανότητα τοποθέτησης, μειώνοντας σημαντικά την υπολογιστική πολυπλοκότητα και επιταχύνοντας τη διαδικασία επίλυσης του παζλ.

9.7.2.4 Έλεγχος Συμβατότητας Ενδείξεων

Ο έλεγχος συμβατότητας ενδείξεων αποτελεί κρίσιμο στάδιο στη μέθοδο `_canCluesFit`, καθώς διασφαλίζει ότι η ένδειξη μπορεί να τοποθετηθεί χωρίς να παραβιάζει τους κανόνες του παζλ. Αφού επαληθευτεί ότι υπάρχει αρκετό μήκος για να τοποθετηθεί η ένδειξη, ο αλγόριθμος εξετάζει αν το τμήμα της λύσης όπου πρόκειται να τοποθετηθεί η ένδειξη είναι συμβατό με τις απαιτήσεις.

Αρχικά, η μέθοδος απομονώνει το τμήμα της λύσης που θα ελεγχθεί, χρησιμοποιώντας τη μέθοδο `sublist`. Το τμήμα αυτό περιλαμβάνει τις θέσεις από το `solutionPosition` έως το `solutionPosition + clue`. Ο πρώτος έλεγχος είναι να διαπιστωθεί αν αυτό το τμήμα περιέχει κάποιο κουτί με την τιμή `0`. Τα κουτιά αυτά υποδεικνύουν ότι δεν ανήκουν σε καμία ένδειξη και επομένως η ένδειξη δεν μπορεί να τοποθετηθεί εκεί. Αντίθετα, αν οι θέσεις περιέχουν `1` ή `?`, ο έλεγχος συνεχίζεται, καθώς αυτές οι τιμές υποδηλώνουν ότι το κουτί μπορεί ενδεχομένως να ανήκει στην ένδειξη.

Στη συνέχεια, γίνεται ένας επιπλέον έλεγχος για τις τιμές πριν και μετά το τμήμα της ένδειξης. Ειδικότερα, ελέγχεται αν η τιμή πριν από το `solutionPosition` ή μετά το `solutionPosition + clue` είναι `1`. Αν ισχύει κάτι τέτοιο, η τοποθέτηση αποτυγχάνει, καθώς αυτό σημαίνει ότι το σύνολο των συνεχόμενων γεμισμένων κουτιών (με τιμή `1`) υπερβαίνει την τιμή της ένδειξης, γεγονός που την καθιστά ασύμβατη με την τοποθέτηση.

Αυτή η λογική αποτυπώνεται στον παρακάτω κώδικα:

```
final List<String> fit = solutionList.sublist(solutionPosition, solutionPosition + clue);

final String valueAfter =
    solutionPosition + clue >= solutionList.length ? '0' : solutionList[solutionPosition + clue];

final String valueBefore =
    solutionPosition <= 0 ? '0' : solutionList[solutionPosition - 1];

final bool canFit = !fit.contains('0') && valueAfter != '1' && valueBefore != '1';
```

Για παράδειγμα, αν η λύση είναι:

```
['?', '1', '?', '0', '?', '1', '0', '?', '?', '?']
```

το `solutionPosition` είναι `2` και το `clue` έχει τιμή `3`, η μέθοδος εξετάζει το τμήμα:

```
['?', '0', '?']
```

Ο πρώτος έλεγχος αποτυγχάνει, καθώς το τμήμα περιέχει τη τιμή `0`, υποδεικνύοντας ότι η ένδειξη δεν μπορεί να τοποθετηθεί σε αυτές τις θέσεις.

Αν το τμήμα περάσει τον πρώτο έλεγχο, τότε ακολουθεί ο έλεγχος των περιφερειακών τιμών. Αν οποιαδήποτε από αυτές είναι `1`, η ένδειξη απορρίπτεται, καθώς αυτό σημαίνει ότι ήδη υπάρχει υπέρβαση στην ακολουθία των γεμισμένων κουτιών σε σχέση με την ένδειξη.

Συνολικά, η διαδικασία αυτή εξασφαλίζει ότι κάθε ένδειξη τοποθετείται με συνέπεια και σύμφωνα με τους κανόνες του παζλ. Ο συνδυασμός των δύο ελέγχων—του τμήματος της ένδειξης και των περιφερειακών τιμών—εξαλείφει περιπτώσεις ασυμβατότητας και ενισχύει την ακρίβεια της λύσης.

9.7.2.4.1 Διαδικασία Ελέγχου

Για να επιτευχθεί αυτός ο στόχος, η μέθοδος καλεί τη βοηθητική συνάρτηση `_doOtherCluesFit` δύο φορές: μία για τις προηγούμενες ενδείξεις και μία για τις επόμενες. Η πρώτη κλήση γίνεται με την παράμετρο `NonoDirection.before` και ελέγχει αν οι προηγούμενες ενδείξεις μπορούν να τοποθετηθούν σωστά μέχρι το σημείο `solutionPosition`. Η δεύτερη κλήση γίνεται με την παράμετρο `NonoDirection.after` και εξετάζει αν οι επόμενες ενδείξεις μπορούν να τοποθετηθούν σωστά μετά το τέλος της τρέχουσας ένδειξης.

Αυτές οι κλήσεις ενσωματώνονται στον κώδικα ως εξής:

```
final bool cluesBeforeGood =
    _doOtherCluesFit(
        NonoDirection.before,
        clues,
        cluePosition,
        solution,
        solutionPosition,
        output,
        settings,
    );

final bool cluesAfterGood =
    _doOtherCluesFit(
        NonoDirection.after,
        clues,
        cluePosition,
        solution,
        solutionPosition,
        output,
        settings,
    );
```

Η μέθοδος `_doOtherCluesFit` υλοποιεί έναν αναδρομικό έλεγχο, εξετάζοντας κάθε ένδειξη στη σειρά. Για τις προηγούμενες ενδείξεις, ο αλγόριθμος προχωρά προς την αρχή της λύσης, ενώ για τις επόμενες, κινείται προς το τέλος. Κατά τη διαδικασία αυτή, εξετάζεται αν κάθε ένδειξη μπορεί να τοποθετηθεί στις αντίστοιχες θέσεις της λύσης, χωρίς να παραβιάζεται η λογική του παζλ.

9.7.2.4.2 Ερμηνεία των Αποτελεσμάτων

Μετά την ολοκλήρωση των δύο κλήσεων, η μέθοδος `_canCluesFit` συνδυάζει τα αποτελέσματα των ελέγχων. Αν και οι δύο επιστρέψουν `true`, τότε η ένδειξη που εξετάζεται μπορεί να τοποθετηθεί στη θέση `solutionPosition`. Αν οποιοσδήποτε από τους δύο ελέγχους αποτύχει, τότε η ένδειξη απορρίπτεται. Η λογική αυτή αποτυπώνεται στον εξής κώδικα:

```
return cluesBeforeGood && cluesAfterGood;
```

Αυτή η συνθήκη εξασφαλίζει ότι τόσο η τρέχουσα ένδειξη όσο και οι υπόλοιπες μπορούν να τοποθετηθούν αρμονικά, δημιουργώντας μια συνεπή και λογική λύση για τη γραμμή ή τη στήλη.

9.7.2.4.3 Σημασία για τη Συνολική Επίλυση

Ο έλεγχος των υπόλοιπων ενδείξεων εξασφαλίζει ότι κάθε τοποθέτηση γίνεται με βάση τη συνολική εικόνα της λύσης, αποφεύγοντας σφάλματα και ασυμβατότητες που θα μπορούσαν να προκύψουν αν η μέθοδος αξιολογούσε μεμονωμένα κάθε ένδειξη. Μέσω της συνεργασίας των `_canCluesFit` και `_doOtherCluesFit`, ο αλγόριθμος αποκτά την απαιτούμενη ακρίβεια για την επιτυχή επίλυση του παζλ.

9.7.2.5 Επιστροφή Αποτελέσματος

Η επιστροφή αποτελέσματος στη μέθοδο `_canCluesFit` είναι το τελικό βήμα, στο οποίο αξιολογείται αν η ένδειξη που εξετάζεται μπορεί να τοποθετηθεί στη συγκεκριμένη θέση της λύσης. Η απόφαση βασίζεται στους ελέγχους που πραγματοποιήθηκαν στα προηγούμενα στάδια: αν η ένδειξη χωράει στο μήκος που απομένει στη λύση, αν είναι συμβατή με τα ήδη γεμισμένα ή κενά κουτάκια, και αν οι υπόλοιπες ενδείξεις μπορούν να τοποθετηθούν χωρίς παραβίαση των κανόνων του παζλ.

9.7.2.6 Διαδικασία Αξιολόγησης

Η μέθοδος συνδυάζει τα αποτελέσματα των ελέγχων των υπόλοιπων ενδείξεων, χρησιμοποιώντας τη λογική `AND` για να διασφαλίσει ότι όλες οι συνθήκες ικανοποιούνται:

```
return cluesBeforeGood && cluesAfterGood;
```

Εδώ, οι δύο τιμές είναι οι εξής:

- `cluesBeforeGood`: Αντιπροσωπεύει το αποτέλεσμα του ελέγχου για τις προηγούμενες ενδείξεις μέσω της μεθόδου `_doOtherCluesFit` με κατεύθυνση `NonoDirection.before`.
- `cluesAfterGood`: Αντιπροσωπεύει το αποτέλεσμα του ελέγχου για τις επόμενες ενδείξεις μέσω της ίδιας μεθόδου με κατεύθυνση `NonoDirection.after`.

Αν και οι δύο τιμές είναι `true`, η μέθοδος επιστρέφει `true`, δηλώνοντας ότι η ένδειξη μπορεί να τοποθετηθεί στη θέση που εξετάζεται. Αν οποιαδήποτε από τις δύο είναι `false`, η μέθοδος επιστρέφει `false`, απορρίπτοντας την τοποθέτηση της ένδειξης.

9.7.2.7 Σημασία της Επιστροφής Αποτελέσματος

Η επιστροφή της τιμής `true` ή `false` στη μέθοδο `_canCluesFit` παρέχει κρίσιμες πληροφορίες για τον συνολικό αλγόριθμο επίλυσης. Αν η ένδειξη μπορεί να τοποθετηθεί, τότε οι επόμενες φάσεις του αλγορίθμου θα συνεχίσουν με αυτή τη θέση ως βάση. Αν όχι, η θέση απορρίπτεται, και ο αλγόριθμος προχωρά στην εξέταση της επόμενης δυνατής τοποθέτησης. Αυτός ο μηχανισμός διασφαλίζει την ακρίβεια και τη συνέπεια της λύσης.

K 9

9.7.3 Έλεγχος τοποθέτησης υπολοίπων στοιχείων (_doOtherCluesFit)

9.7.3.1 Εισαγωγή

Η μέθοδος _doOtherCluesFit αποτελεί μια από τις πιο σημαντικές συνιστώσες του αλγορίθμου επίλυσης του παζλ εικονόσταυρου, καθώς εξετάζει αν οι υπόλοιπες ενδείξεις (πριν ή μετά από την τρέχουσα ένδειξη) μπορούν να τοποθετηθούν στη λύση. Μέσα από μια σειρά αναλυτικών ελέγχων και αλληλεπιδράσεων με τη μέθοδο _canCluesFit, η μέθοδος αυτή εξασφαλίζει ότι κάθε ένδειξη προσαρμόζεται με συνέπεια στο διαθέσιμο χώρο, χωρίς να παραβιάζονται οι περιορισμοί του παζλ.

9.7.3.2 Εισαγωγή στη Μέθοδο

Η _doOtherCluesFit εστιάζει στον έλεγχο των υπολοίπων ενδείξεων είτε πριν είτε μετά την τρέχουσα ένδειξη, με βάση την κατεύθυνση που ορίζεται μέσω της παραμέτρου solutionSide (τύπου NonoDirection). Ανάλογα με τη θέση της ένδειξης στη λίστα των ενδείξεων (clueIndex), η μέθοδος ελέγχει:

- Αν υπάρχουν άλλες ενδείξεις να τοποθετηθούν.
- Αν υπάρχει αρκετός διαθέσιμος χώρος.
- Αν η λύση σε αυτό το υποσύνολο μπορεί να ολοκληρωθεί σωστά.

9.7.3.3 Ρύθμιση και Ενημέρωση Στατιστικών

Στην αρχή της μεθόδου, ενημερώνεται ο μετρητής που καταγράφει τον αριθμό των κουτιών που έχουν ελεγχθεί κατά την επεξεργασία, αν η συγκεκριμένη δυνατότητα είναι ενεργοποιημένη μέσω του SolverSettings:

```
if (settings.countCheckedBoxes) {
  output.otherBoxesCheckedList
    ..add(output.otherBoxesCheckedList.last + 1)
    ..removeAt(0);
}
```

Αυτή η λειτουργικότητα επιτρέπει τη μέτρηση των επαναλήψεων και ελέγχων που πραγματοποιούνται κατά την επίλυση του παζλ.

9.7.3.4 Έλεγχος για Υπόλοιπες Ενδείξεις

Ο έλεγχος για το αν υπάρχουν ενδείξεις πριν ή μετά από την τρέχουσα πραγματοποιείται μέσω της μεθόδου hasOtherClues της κλάσης NonoDirection. Αν δεν υπάρχουν άλλες ενδείξεις, η μέθοδος ελέγχει αν η λύση στη συγκεκριμένη θέση είναι έγκυρη μέσω της isSolutionValid:

```
if (!solutionSide.hasOtherClues(clueIndex, clues.length)) {
  return solutionSide.isSolutionValid(solution, solutionIndex, clues[clueIndex]);
}
```

9.7.3.5 Δημιουργία Υπολίστας Ενδείξεων

Αν υπάρχουν υπόλοιπες ενδείξεις, δημιουργείται μια υπολίστα αυτών μέσω της `getCluesSublist`, η οποία περιλαμβάνει τις σχετικές ενδείξεις:

```
final List<int> cluesSublist = solutionSide.getCluesSublist(clueIndex, clues);
```

Η υπολίστα αυτή περιλαμβάνει όλες τις ενδείξεις πριν ή μετά την τρέχουσα ένδειξη, ανάλογα με την κατεύθυνση που ορίζεται.

9.7.3.6 Έλεγχος Επαρκούς Χώρου

Η μέθοδος ελέγχει αν υπάρχει αρκετός διαθέσιμος χώρος στη λύση για την τοποθέτηση των υπόλοιπων ενδείξεων, χρησιμοποιώντας τη `hasBoxesLeft`. Αν δεν υπάρχει επαρκής χώρος, η μέθοδος επιστρέφει `false`:

```
if (!solutionSide.hasBoxesLeft(solutionIndex, clue, solution, cluesSublist)) {
  return false;
}
```

Αυτός ο έλεγχος είναι κρίσιμος, καθώς αποτρέπει την περιττή επεξεργασία υποθέσεων που είναι προφανώς αδύνατες λόγω έλλειψης χώρου.

9.7.3.7 Αναδρομικός Έλεγχος Υπολίστας Λύσης

Η μέθοδος δημιουργεί μια υπολίστα της λύσης, περιλαμβάνοντας τον διαθέσιμο χώρο μετά την τρέχουσα ένδειξη, και καλεί αναδρομικά τη `_canCluesFit`:

```
final String solutionSublist =
    solutionSide.getSolutionSublist(solution, solutionIndex, clue);

for (int solutionSublistIndex = 0;
    solutionSublistIndex < solutionSublist.length;
    solutionSublistIndex++) {
  if (_canCluesFit(cluesSublist, solutionSublist,
                   solutionSublistIndex, 0,
                   output, settings)) {
    return true;
  }
}
```

9.7.3.8 Επιστροφή Αποτελέσματος

Αν καμία από τις υποθέσεις δεν οδηγήσει σε έγκυρη λύση, η μέθοδος επιστρέφει:

```
return false;
```

Αυτό σημαίνει ότι οι υπόλοιπες ενδείξεις δεν μπορούν να τοποθετηθούν σωστά στη λύση, και η τρέχουσα υπόθεση απορρίπτεται.

9.7.3.9 Συμπέρασμα

Η `_doOtherCluesFit` είναι ζωτικής σημασίας για την ακεραιότητα και την ακρίβεια του αλγορίθμου επίλυσης. Μέσα από λεπτομερείς ελέγχους και αναδρομικές κλήσεις, διασφαλίζει ότι κάθε ένδειξη και κάθε θέση εξετάζονται στο κατάλληλο πλαίσιο, παρέχοντας ένα ευέλικτο εργαλείο για τη διαχείριση πολύπλοκων σχέσεων μεταξύ των ενδείξεων και του διαθέσιμου χώρου στη λύση.

9.8 Αρχείο Υποστηρικτικών Συναρτήσεων

9.8.1 Εισαγωγή

Το αρχείο `LineSolverHelper` περιλαμβάνει κρίσιμες υποστηρικτικές συναρτήσεις που ενισχύουν τη λειτουργία των κύριων αλγορίθμων επίλυσης γραμμών και στηλών. Αυτές οι συναρτήσεις εκτελούν εργασίες όπως η δημιουργία της λίστας (σωρού), η ανάκτηση δεδομένων γραμμών και στηλών, ο εντοπισμός μη συμπληρωμένων κουτιών, η ενημέρωση της λύσης και η διαχείριση της μνήμης cache. Η παρακάτω ανάλυση εστιάζει στη λειτουργικότητα καθεμίας από αυτές τις συναρτήσεις, με παραδείγματα και εξηγήσεις για τον τρόπο με τον οποίο ο κώδικας υλοποιεί αυτές τις λειτουργίες. Παρουσιάζονται οι βασικές λειτουργίες του αρχείου `LineSolverHelper`, καθώς και ο ρόλος τους στη συνολική διαδικασία.

9.8.2 Δημιουργία Λίστας (Σωρού) (`initializeStackList`)

9.8.2.1 Εισαγωγή

Η μέθοδος `initializeStackList` διαδραματίζει σημαντικό ρόλο στην αρχική διαμόρφωση της λίστας σωρού (`stack`) που χρησιμοποιείται κατά την επίλυση παζλ εικονόσταυρου (`nonogram`). Ο στόχος της μεθόδου είναι να δημιουργήσει μια λίστα που περιλαμβάνει όλους τους δείκτες γραμμών και στηλών του παζλ, καθιστώντας τους έτοιμους για επεξεργασία. Παράλληλα, παρέχεται η δυνατότητα ταξινόμησης των γραμμών και στηλών με βάση το άθροισμα των ενδείξεών τους, ώστε να δοθεί προτεραιότητα στις πιο «πλούσιες» σε πληροφορία γραμμές ή στήλες.

9.8.2.2 Τι κάνει η μέθοδος

Η `initializeStackList` δέχεται ως είσοδο ένα αντικείμενο `Clues`, το οποίο περιέχει τις ενδείξεις (`clues`) για όλες τις γραμμές και στήλες του παζλ, και μια προαιρετική λογική για ταξινόμηση. Εάν επιλεγεί η ταξινόμηση (`sortInitialLinesStackViaClues = true`), τότε η μέθοδος ταξινομεί τις γραμμές και τις στήλες κατά φθίνουσα σειρά του αθροίσματος των ενδείξεών τους. Εάν δεν επιλεγεί, δημιουργεί απλώς έναν μη ταξινομημένο σωρό με όλα τα στοιχεία. Η μέθοδος επιστρέφει μια λίστα από χάρτες (`List<Map<int, NonoAxis>>`), όπου κάθε χάρτης περιλαμβάνει τη θέση (`index`) μιας γραμμής ή στήλης και τον αντίστοιχο άξονα (`NonoAxis.row` ή `NonoAxis.column`).

9.8.2.3 Πώς λειτουργεί η μέθοδος

Η μέθοδος υλοποιείται σε δύο βασικά στάδια: τη μη ταξινομημένη δημιουργία και την ταξινομημένη δημιουργία.

9.8.2.3.1 Μη ταξινομημένη δημιουργία σωρού

Αν η παράμετρος `sortInitialLinesStackViaClues` είναι `false`, η μέθοδος διατρέχει όλες τις γραμμές και στήλες και δημιουργεί ένα χάρτη για καθεμία. Ο χάρτης περιλαμβάνει το δείκτη (`index`) και τον τύπο του άξονα.

Ο κώδικας για αυτό είναι:

```
return <Map<int, NonoAxis>>[
  for (int i = 0; i < clues.rows.length; i++) <int, NonoAxis>{i: NonoAxis.row},
  for (int i = 0; i < clues.columns.length; i++) <int, NonoAxis>{i: NonoAxis.column},
];
```

- Η `clues.rows.length` καθορίζει τον αριθμό των γραμμών.
- Η `clues.columns.length` καθορίζει τον αριθμό των στηλών.
- Οι γραμμές και οι στήλες προστίθενται διαδοχικά στη λίστα.

9.8.2.3.2 Ταξινομημένη δημιουργία σωρού

Αν η παράμετρος `sortInitialLinesStackViaClues` είναι `true`, η μέθοδος πρώτα υπολογίζει το άθροισμα των ενδείξεων για κάθε γραμμή και στήλη, δημιουργώντας μια προσωρινή λίστα από χάρτες που περιλαμβάνουν το άθροισμα των ενδείξεων ως κλειδί. Ο κώδικας είναι:

```
final List<Map<String, NonoAxis>> map = <Map<String, NonoAxis>>[
  for (int i = 0; i < clues.rows.length; i++)
    <String, NonoAxis>{'$i,${clues.rows[i].sum}': NonoAxis.row},
  for (int i = 0; i < clues.columns.length; i++)
    <String, NonoAxis>{'$i,${clues.columns[i].sum}': NonoAxis.column},
];
```

- Η μέθοδος `sum` υπολογίζει το άθροισμα των ενδείξεων για κάθε γραμμή ή στήλη.
- Το αποτέλεσμα αποθηκεύεται με τη μορφή `'$index,${sum}'`.

Στη συνέχεια, η λίστα ταξινομείται κατά φθίνουσα σειρά με βάση το άθροισμα:

```
map.sort(
  (Map<String, NonoAxis> a, Map<String, NonoAxis> b) {
    final int aClueSum = int.parse(a.keys.first.split(',')[1]);
    final int bClueSum = int.parse(b.keys.first.split(',')[1]);
    return bClueSum.compareTo(aClueSum);
  },
);
```

Τέλος, η ταξινομημένη λίστα μετατρέπεται σε μια λίστα με χάρτες που περιλαμβάνουν μόνο το δείκτη και τον τύπο άξονα:

```
return map.map((Map<String, NonoAxis> e) =>
  <int, NonoAxis>{int.parse(e.keys.first.split(',')[0]): e.values.first}).toList();
```

9.8.2.4 Παράδειγμα χρήσης

Αν το παζλ έχει τις εξής ενδείξεις:

- Γραμμές: `[[5], [1]]`
- Στήλες: `[[2, 2], [2]]`

Χωρίς ταξινόμηση (`sortInitialLinesStackViaClues = false`), η μέθοδος επιστρέφει:

```
[0: NonoAxis.row, 1: NonoAxis.row, 0: NonoAxis.column, 1: NonoAxis.column]
```

Με ταξινόμηση (`sortInitialLinesStackViaClues = true`), η μέθοδος επιστρέφει:

```
[0: NonoAxis.row, 0: NonoAxis.column, 1: NonoAxis.column, 1: NonoAxis.row]
```

Αυτό δείχνει ότι οι γραμμές και οι στήλες με τα μεγαλύτερα αθροίσματα ενδείξεων λαμβάνουν προτεραιότητα στον σωρό.

9.8.3 Ανάκτηση Κατάστασης Γραμμής ή Στήλης (`getSolutionLine`)

9.8.3.1 Εισαγωγή

Η μέθοδος `getSolutionLine` είναι υπεύθυνη για την ανάκτηση της τρέχουσας κατάστασης μιας γραμμής ή στήλης από τη συνολική λύση του παζλ εικονόσταυρου. Αυτή η λειτουργία είναι θεμελιώδης για τη διαχείριση της κατάστασης του παζλ, καθώς επιτρέπει στον αλγόριθμο να επικεντρώνεται σε συγκεκριμένα τμήματα της λύσης, όπως γραμμές ή στήλες, προκειμένου να εφαρμόσει τη λογική της επίλυσης. Η μέθοδος υποστηρίζει τόσο οριζόντιους (γραμμές) όσο και κάθετους (στήλες) άξονες και επιστρέφει την τρέχουσα κατάσταση της ζητούμενης γραμμής ή στήλης με τη μορφή συμβολοσειράς.

9.8.3.2 Τι κάνει η μέθοδος

Η `getSolutionLine` δέχεται ως είσοδο:

- Την πλήρη λύση του παζλ (μια συμβολοσειρά),
- Το πλάτος του παζλ (`nonogramWidth`),
- Τον δείκτη της γραμμής ή της στήλης (`lineIndex`),
- Τον τύπο του άξονα (`NonoAxis.row` ή `NonoAxis.column`).

Με βάση τον τύπο του άξονα, εφαρμόζει διαφορετική λογική για να απομονώσει είτε τα στοιχεία μιας γραμμής είτε μιας στήλης, επιστρέφοντας μια συμβολοσειρά που περιλαμβάνει μόνο τα κουτιά της συγκεκριμένης γραμμής ή στήλης.

9.8.3.3 Πώς λειτουργεί η μέθοδος

9.8.3.3.1 Ανάκτηση κατάστασης γραμμής (`NonoAxis.row`)

Όταν ο τύπος του άξονα είναι `NonoAxis.row`, η μέθοδος υπολογίζει το εύρος των χαρακτήρων που αντιστοιχούν στη γραμμή με τα παρακάτω βήματα:

1. Χωρίζει τη συνολική λύση σε πίνακα χαρακτήρων μέσω της `split('')`.
2. Υπολογίζει το εύρος των χαρακτήρων της γραμμής:

   ```
   solutionList.getRange(lineIndex * nonogramWidth, nonogramWidth * (lineIndex + 1))
   ```

3. Επανασυνθέτει τους χαρακτήρες σε συμβολοσειρά μέσω της `join()`.

Παράδειγμα: Αν η συνολική λύση είναι `"123456789"`, το πλάτος του παζλ είναι `3`, και ο δείκτης της γραμμής είναι `1`, τότε:

- Το εύρος είναι από 3 έως 6.
- Το αποτέλεσμα είναι η συμβολοσειρά `"456"`.

9.8.3.3.2 Ανάκτηση κατάστασης στήλης (`NonoAxis.column`)

Όταν ο τύπος του άξονα είναι `NonoAxis.column`, η μέθοδος συλλέγει τους χαρακτήρες που ανήκουν στη στήλη, χρησιμοποιώντας έναν επαναληπτικό βρόχο:

```
for (int solChar = lineIndex; solChar < solution.length; solChar += nonogramWidth) {
  columnSol += solution[solChar];
}
```

Παράδειγμα: Αν η συνολική λύση είναι `"123456789"`, το πλάτος του παζλ είναι `3`, και ο δείκτης της στήλης είναι `1`, τότε:

- Οι θέσεις είναι 1, 4, 7.
- Το αποτέλεσμα είναι η συμβολοσειρά `"258"`.

9.8.3.4 Παραδείγματα Χρήσης της `getSolutionLine`

Η μέθοδος `getSolutionLine` μπορεί να χρησιμοποιηθεί για την ανάκτηση είτε μιας γραμμής είτε μιας στήλης από τη συνολική λύση του παζλ, ανάλογα με τον τύπο του άξονα (`NonoAxis.row` ή `NonoAxis.column`).

9.8.3.4.1 Ανάκτηση Γραμμής

Για την περίπτωση μιας γραμμής, η μέθοδος χρησιμοποιεί το πλάτος του παζλ για να υπολογίσει το εύρος των χαρακτήρων που αντιστοιχούν στη συγκεκριμένη γραμμή.

Παράδειγμα:

- Συνολική λύση: `"123456789"`
- Πλάτος παζλ: `3`
- Δείκτης γραμμής (`lineIndex`): `1` (δεύτερη γραμμή)

Η μέθοδος επιστρέφει τη συμβολοσειρά `"456"`, η οποία αντιπροσωπεύει τους χαρακτήρες της δεύτερης γραμμής.

9.8.3.4.2 Ανάκτηση Στήλης

Για την ανάκτηση μιας στήλης, η μέθοδος συλλέγει τους χαρακτήρες από κάθε γραμμή που αντιστοιχούν στη ζητούμενη στήλη, χρησιμοποιώντας έναν επαναληπτικό βρόχο.

Παράδειγμα:

- Συνολική λύση: `"123456789"`
- Πλάτος παζλ: `3`
- Δείκτης στήλης (`lineIndex`): `1` (δεύτερη στήλη)

Η μέθοδος επιστρέφει τη συμβολοσειρά `"258"`, η οποία αντιστοιχεί στους δεύτερους χαρακτήρες κάθε γραμμής.

9.8.3.4.3 Συνοπτική Περιγραφή

Αυτά τα παραδείγματα δείχνουν πώς η `getSolutionLine` προσαρμόζεται για να ανακτήσει είτε μια γραμμή είτε μια στήλη με ακρίβεια, επιτρέποντας την περαιτέρω επεξεργασία της λύσης του παζλ. Είτε πρόκειται για οριζόντια είτε για κάθετη ανάκτηση, η μέθοδος παρέχει έναν σαφή και αποδοτικό τρόπο πρόσβασης στα δεδομένα του παζλ.

9.8.4 Εντοπισμός Μη Συμπληρωμένων Κουτιών (`getCharIndexesOfQuestionMarks`)

9.8.4.1 Εισαγωγή

Η μέθοδος `getCharIndexesOfQuestionMarks` έχει ως στόχο να εντοπίσει τις θέσεις των μη συμπληρωμένων κουτιών (που αντιπροσωπεύονται από τον χαρακτήρα `?`) σε μια συμβολοσειρά που αντιπροσωπεύει την τρέχουσα λύση μιας γραμμής ή στήλης ενός παζλ εικονόσταυρου. Η μέθοδος επιστρέφει μια λίστα με τους δείκτες (`List<int>`) όπου βρίσκονται οι χαρακτήρες `?` στη συμβολοσειρά, διευκολύνοντας έτσι τον εντοπισμό των θέσεων που χρειάζονται περαιτέρω επεξεργασία ή συμπλήρωση.

9.8.4.2 Περιγραφή Λειτουργίας

Η λειτουργία της μεθόδου περιλαμβάνει τα εξής στάδια:

- **Διάσπαση Συμβολοσειράς:** Η συμβολοσειρά της λύσης διασπάται σε μεμονωμένους χαρακτήρες μαζί με τους δείκτες τους, χρησιμοποιώντας την επέκταση `indexed`. Αυτό δημιουργεί μια λίστα ζευγών όπου κάθε ζεύγος περιέχει τη θέση και τον χαρακτήρα. **Παράδειγμα:** Αν η αρχική συμβολοσειρά είναι `"0?1"`, η `indexed` δημιουργεί:

```
[(0, '0'), (1, '?'), (2, '1')]
```

- **Χρήση Κανονικής Έκφρασης:** Η λίστα μετατρέπεται σε συμβολοσειρά μέσω της `toString`, και ακολουθεί η χρήση μιας κανονικής έκφρασης (Regular Expression) για τον εντοπισμό των θέσεων των χαρακτήρων `?`. Η κανονική έκφραση που χρησιμοποιείται είναι:

  ```
  RegExp charIndexesRegexp = RegExp(r'[0-9]+(?=, \?)');
  ```

  Αυτή η έκφραση εντοπίζει αριθμούς (`[0-9]+`) που ακολουθούνται από τη συμβολοσειρά `", ?"`.

- **Εξαγωγή και Μετατροπή:** Μετά την εύρεση όλων των αντιστοιχιών, οι αριθμοί εξάγονται, ενώνονται σε μια νέα συμβολοσειρά χωρισμένη με κόμματα, και στη συνέχεια μετατρέπονται ξανά σε ακέραιους (`int`) για να δημιουργηθεί η τελική λίστα των δεικτών.

9.8.4.3 Παραδείγματα

Ας υποθέσουμε ότι η είσοδος της μεθόδου είναι η συμβολοσειρά `"0?10?"`. Η διαδικασία εξελίσσεται ως εξής:

1. **Διάσπαση και Επεξεργασία:** Η συμβολοσειρά `"0?10?"` μετατρέπεται σε ζεύγη μέσω `indexed`:

   ```
   [(0, '0'), (1, '?'), (2, '1'), (3, '0'), (4, '?')]
   ```

2. **Εντοπισμός με Κανονική Έκφραση:** Η κανονική έκφραση εντοπίζει τους δείκτες `1` και `4` που αντιστοιχούν στους χαρακτήρες `?`.

3. **Μετατροπή σε Λίστα:** Οι δείκτες επιστρέφονται ως λίστα:

   ```
   [1, 4]
   ```

Για παράδειγμα, αν η είσοδος της μεθόδου είναι `"((0, 0), (1, ?), (2, 1))"`, η μέθοδος επιστρέφει:

```
[1]
```

9.8.4.4 Συμπέρασμα

Η μέθοδος `getCharIndexesOfQuestionMarks` είναι κρίσιμη για την αποδοτική ανάλυση των μη συμπληρωμένων περιοχών ενός παζλ, καθιστώντας την απαραίτητη για την περαιτέρω επεξεργασία της λύσης.

9.8.5 Ενημέρωση Λύσης (`getFilledInSolution`)

9.8.5.1 Εισαγωγή

Η μέθοδος `getFilledInSolution` παίζει κρίσιμο ρόλο στη διαδικασία επίλυσης ενός παζλ εικονόσταυρου. Στόχος της είναι να ενημερώσει τη συμβολοσειρά της λύσης αντικαθιστώντας συγκεκριμένους χαρακτήρες με 0 ή 1, ανάλογα με την πρόοδο που έχει σημειωθεί. Η μέθοδος χρησιμοποιείται για να διασφαλίσει ότι η λύση αντικατοπτρίζει με ακρίβεια τα κουτιά που έχουν επιβεβαιωθεί είτε ως γεμισμένα (1) είτε ως διαγραμμένα (0). Παράλληλα, δημιουργεί μια λίστα με τις θέσεις των νέων στοιχείων που έχουν ενημερωθεί, η οποία χρησιμοποιείται για να εμφανιστεί οπτικά στον χρήστη ποια κουτιά έχουν αλλάξει, χρωματίζοντάς τα με διαφορετικό χρώμα (πράσινο αντί για μαύρο).

9.8.5.2 Πώς Λειτουργεί η Μέθοδος

Η μέθοδος εκτελείται ως εξής:

- **Αρχικοποίηση:** Η μέθοδος ξεκινά με την αρχικοποίηση της συμβολοσειράς `fullUpdatedSolution`, που περιέχει την τρέχουσα λύση, και μιας λίστας `newFilledBoxes`, που θα περιέχει τις θέσεις των κουτιών που ενημερώνονται.
- **Επεξεργασία Θέσεων:** Σε κάθε επανάληψη, η μέθοδος καλεί τη `getSolutionPosition` για να υπολογιστεί η ακριβής θέση του χαρακτήρα που πρέπει να αντικατασταθεί:

  ```
  final int tempPos = lineType.getSolutionPosition(lineIndex, charIndex, nonogramWidth);
  ```

- **Αντικατάσταση Χαρακτήρα:** Ο χαρακτήρας στη συγκεκριμένη θέση αντικαθίσταται με `0` ή `1`, ανάλογα με την τιμή του `clueKey`:

  ```
  fullUpdatedSolution = '${fullUpdatedSolution.substring(0, tempPos)}'
                        '${clueKey == 0 ? '0' : '1'}'
                        '${fullUpdatedSolution.substring(tempPos + 1)}';
  ```

- **Ενημέρωση Λίστας:** Παράλληλα, η θέση `tempPos` προστίθεται στη λίστα `newFilledBoxes`. Αυτή η λίστα χρησιμοποιείται για να επισημάνει οπτικά τα κουτιά που ενημερώθηκαν, χρωματίζοντάς τα πράσινα στην οθόνη.
- **Επιστροφή Αποτελέσματος:** Στο τέλος, η μέθοδος επιστρέφει έναν χάρτη (`Map`) που περιέχει:
  - Την ενημερωμένη συμβολοσειρά της λύσης (`fullUpdatedSolution`).
  - Τη λίστα με τις θέσεις των νέων στοιχείων (`newFilledBoxes`).

9.8.5.3 Παραδείγματα Χρήσης

Για να κατανοηθεί καλύτερα η μέθοδος, ας δούμε ένα παράδειγμα.

Δεδομένα:

- Τρέχουσα λύση: `"010?000?1??0"`
- Δείκτης γραμμής: `1`
- Τύπος άξονα: `NonoAxis.row`
- Πλάτος παζλ: `6`
- Δείκτες χαρακτήρων: `[1, 3]`
- `ClueKey`: `1`

Διαδικασία:

1. **Υπολογισμός Θέσεων:** Οι θέσεις που πρέπει να ενημερωθούν υπολογίζονται:

   ```
   final int tempPos = lineType.getSolutionPosition(lineIndex, charIndex, nonogramWidth);
   ```

   Για το παραπάνω παράδειγμα, οι θέσεις είναι `7` και `9`.

2. **Αντικατάσταση Χαρακτήρων:** Οι χαρακτήρες στις θέσεις 7 και 9 αντικαθίστανται με 1, δημιουργώντας τη νέα λύση:

   "010?000111?0"

3. **Επιστροφή Αποτελέσματος:** Η μέθοδος επιστρέφει:

   - `fullUpdatedSolution: "010?000111?0"`
   - `newFilledBoxes: [7, 9]`

9.8.5.4 Συμπέρασμα

Η μέθοδος `getFilledInSolution` διασφαλίζει ότι η λύση του παζλ αντικατοπτρίζει τις τελευταίες ενημερώσεις με ακρίβεια. Η δυνατότητα να επισημαίνονται οι αλλαγές οπτικά βοηθά τους χρήστες να κατανοήσουν καλύτερα τη λογική πίσω από τη λύση και την πρόοδο του αλγορίθμου.

9.8.6 Διαχείριση Μνήμης Cache (`updateCachedBoxSolutions`)

9.8.6.1 Εισαγωγή

Η μέθοδος `updateCachedBoxSolutions` χρησιμοποιείται για τη διαχείριση δεδομένων στη μνήμη *cache*, με σκοπό τη βελτιστοποίηση της διαδικασίας επίλυσης του παζλ εικονόσταυρου. Συγκεκριμένα, αποθηκεύει το αποτέλεσμα της μεθόδου `_canCluesFit` (αναλυτικά στην ενότητα **??**) για έναν συγκεκριμένο συνδυασμό ενδείξεων, θέσης και κατάστασης λύσης. Το αποτέλεσμα αυτό υποδεικνύει αν μια συγκεκριμένη ένδειξη (`clueIndex`) μπορεί να τοποθετηθεί σε μια καθορισμένη θέση (`solutionIndex`) μιας γραμμής ή στήλης, λαμβάνοντας υπόψη την τρέχουσα κατάσταση της λύσης.

Η χρήση της *cache* βοηθά στη μείωση επαναλαμβανόμενων υπολογισμών, παρέχοντας γρήγορη πρόσβαση σε προϋπάρχουσες πληροφορίες. Παρόλο που η *cache* προσφέρει σημαντικά οφέλη, απαιτεί προσεκτική διαχείριση για να διασφαλιστεί η αποδοτικότητα του αλγορίθμου.

9.8.6.2 Πώς Λειτουργεί η Μέθοδος

Η μέθοδος `updateCachedBoxSolutions` λαμβάνει τα εξής ορίσματα:

- `clues`: Λίστα ακέραιων αριθμών που αντιπροσωπεύουν τις ενδείξεις για τη γραμμή ή τη στήλη.
- `clueIndex`: Ο δείκτης της ένδειξης που εξετάζεται.
- `solution`: Η τρέχουσα κατάσταση της γραμμής ή στήλης, εκφρασμένη ως συμβολοσειρά.
- `solutionIndex`: Η θέση στη γραμμή ή στήλη όπου εξετάζεται η τοποθέτηση της ένδειξης.
- `canFit`: Το αποτέλεσμα της `_canCluesFit`, που δηλώνει αν η ένδειξη μπορεί να τοποθετηθεί στη συγκεκριμένη θέση (`true/false`).

Η μέθοδος δημιουργεί ένα μοναδικό κλειδί (`key`) που περιγράφει πλήρως την τρέχουσα κατάσταση:

```
'$clues,$clueIndex,$solution,$solutionIndex'
```

Αυτό το key, μαζί με την τιμή canFit, αποθηκεύεται στη μνήμη *cache* υπό τη μορφή χάρτη (Map<String, bool>). Το αποτέλεσμα επιστρέφεται ως εξής:

```
return <String, bool>{'$clues,$clueIndex,$solution,$solutionIndex': canFit};
```

Αυτή η διαδικασία διασφαλίζει ότι, αν ο αλγόριθμος χρειαστεί ξανά να αξιολογήσει την ίδια συνθήκη, μπορεί να ανακτήσει το αποθηκευμένο αποτέλεσμα άμεσα, αποφεύγοντας περιττούς υπολογισμούς.

9.8.6.3 Παραδείγματα Χρήσης

Παράδειγμα Εισόδου:

- clues: [3, 2]
- clueIndex: 1
- solution: "??1??"
- solutionIndex: 2
- canFit: true (αποτέλεσμα της _canCluesFit)

Η μέθοδος δημιουργεί το ακόλουθο κλειδί:

```
'[3, 2],1,??1??,2'
```

Το αποτέλεσμα αποθηκεύεται στη μνήμη *cache*:

```
{'$clues,$clueIndex,$solution,$solutionIndex' -> '[3, 2],1,??1??,2': true}
```

Επανεξέταση: Αν ο αλγόριθμος χρειαστεί ξανά να επεξεργαστεί τη γραμμή ή μια άλλη γραμμή με τα ίδια δεδομένα, θα χρησιμοποιήσει την τιμή true από τη *cache*, παρακάμπτοντας την εκτέλεση της _canCluesFit.

9.8.6.4 Συμπέρασμα

Η μέθοδος updateCachedBoxSolutions επιτρέπει την αποδοτική διαχείριση της μνήμης *cache*, ενισχύοντας τη συνολική απόδοση του αλγορίθμου επίλυσης. Η αποθήκευση αποτελεσμάτων αποτρέπει την επανάληψη χρονοβόρων ελέγχων, ενώ παράλληλα μειώνει το συνολικό κόστος επεξεργασίας, ιδιαίτερα σε μεγάλα ή σύνθετα παζλ. Ωστόσο, η σωστή διαχείριση της *cache* είναι κρίσιμη για την εξασφάλιση της ισορροπίας μεταξύ απόδοσης και κατανάλωσης μνήμης.

9.9 Αρχείο με Τύπους και τις Προεκτάσεις τους

9.9.1 Εισαγωγή

Το αρχείο αυτό περιλαμβάνει όλους τους τύπους δεδομένων και τις επεκτάσεις που συνοδεύουν αυτούς τους τύπους, καθώς και τους βασικούς τύπους που χρησιμοποιούνται στο πλαίσιο της Flutter. Στόχος είναι η τυποποίηση των λειτουργιών και η διευκόλυνση της επεξεργασίας δεδομένων στο πλαίσιο των παζλ εικονόσταυρου.

9.9.2 Ορισμός και Χρήση της `NonoAxisAlignment`

9.9.2.1 Εισαγωγή

Η `NonoAxisAlignment` είναι μια απαρίθμηση (*enumeration*) που προσφέρει επιλογές ευθυγράμμισης για τους άξονες ενός παζλ εικονόσταυρου. Χρησιμοποιείται για την καθοδήγηση της κατεύθυνσης επεξεργασίας ενδείξεων σε έναν άξονα, είτε από την αρχή (`start`) είτε από το τέλος (`end`) αυτού.

9.9.2.2 Ευθυγράμμιση στην Αρχή (`start`)

Η επιλογή `start` χρησιμοποιείται για να ξεκινήσει η επεξεργασία από την αρχή ενός άξονα:

- Για γραμμές: Από τα αριστερά προς τα δεξιά.
- Για στήλες: Από πάνω προς τα κάτω.

Παράδειγμα χρήσης:

```
NonoAxisAlignment alignment = NonoAxisAlignment.start;
```

9.9.2.3 Ευθυγράμμιση στο Τέλος (`end`)

Η επιλογή `end` χρησιμοποιείται για να ξεκινήσει η επεξεργασία από το τέλος ενός άξονα:

- Για γραμμές: Από τα δεξιά προς τα αριστερά.
- Για στήλες: Από κάτω προς τα πάνω.

Παράδειγμα χρήσης:

```
NonoAxisAlignment alignment = NonoAxisAlignment.end;
```

9.9.2.4 Σύνοψη

Η απλότητα της `NonoAxisAlignment` διευκολύνει τη χρήση της στον κώδικα, ενισχύοντας τη σαφήνεια και την αναγνωσιμότητα των λειτουργιών επεξεργασίας.

9.9.3 Ορισμός και Επεξεργασία Άξονα μέσω της `NonoAxis`

9.9.3.1 Εισαγωγή

Η `NonoAxis` είναι μια απαρίθμηση (*enumeration*) που ορίζει τους δύο βασικούς άξονες ενός παζλ εικονόσταυρου: τον οριζόντιο άξονα (γραμμές - `row`) και τον κάθετο άξονα (στήλες - `column`). Αυτή η απαρίθμηση αποτελεί τη βάση για τον καθορισμό του τρόπου επεξεργασίας των δεδομένων του παζλ, καθώς διαχωρίζει τη λογική που εφαρμόζεται στις γραμμές από εκείνη που εφαρμόζεται στις στήλες.

Οι δύο επιλογές της `NonoAxis` είναι:

- `row`: Αντιπροσωπεύει τον οριζόντιο άξονα, δηλαδή τις γραμμές του παζλ.
- `column`: Αντιπροσωπεύει τον κάθετο άξονα, δηλαδή τις στήλες του παζλ.

Αυτή η διάκριση είναι θεμελιώδης για τον αλγόριθμο, καθώς καθορίζει τον τρόπο με τον οποίο προσεγγίζονται οι θέσεις των κουτιών, οι ενδείξεις (*clues*) και οι διαδικασίες συμπλήρωσης ή διαγραφής.

9.9.3.2 Υπολογισμός Θέσης μέσω της `getSolutionPosition`

9.9.3.2.1 Εισαγωγή

Η `getSolutionPosition` είναι μια μέθοδος που περιλαμβάνεται στην επέκταση της απαρίθμησης `NonoAxis`. Χρησιμεύει στον υπολογισμό της θέσης ενός στοιχείου μέσα στη συνολική λύση του παζλ, ανάλογα με το αν πρόκειται για γραμμή (`row`) ή στήλη (`column`).

9.9.3.2.2 Περιγραφή Λειτουργίας

Η `getSolutionPosition` δέχεται τρεις παραμέτρους:

- `lineIndex`: Ο δείκτης της γραμμής ή στήλης.
- `charIndex`: Ο δείκτης του χαρακτήρα μέσα στη γραμμή ή στήλη.
- `nonoWidth`: Το πλάτος του παζλ εικονόσταυρου.

Ανάλογα με τον άξονα (`row` ή `column`), υπολογίζει τη θέση του χαρακτήρα στη συνολική λύση.

9.9.3.2.3 Υπολογισμός Θέσης για Γραμμές

Για γραμμές, η θέση υπολογίζεται ως:

$$\text{Θέση} = \text{lineIndex} \times \text{nonoWidth} + \text{charIndex}$$

Παράδειγμα:

- `lineIndex = 2`, `nonoWidth = 5`, `charIndex = 3`
- Θέση = $2 \times 5 + 3 = 13$

9.9.3.2.4 Υπολογισμός Θέσης για Στήλες

Για στήλες, η θέση υπολογίζεται ως:

$$\text{Θέση} = \text{charIndex} \times \text{nonoWidth} + \text{lineIndex}$$

Παράδειγμα:

- `lineIndex = 2`, `nonoWidth = 5`, `charIndex = 3`
- Θέση = $3 \times 5 + 2 = 17$

9.9.3.2.5 Χρησιμότητα της `getSolutionPosition`

Η μέθοδος `getSolutionPosition` παρέχει έναν τυποποιημένο και αποδοτικό τρόπο για την εύρεση των θέσεων των στοιχείων στη συνολική λύση. Είναι ιδιαίτερα χρήσιμη σε περιπτώσεις που η επεξεργασία των στοιχείων πρέπει να λάβει υπόψη τη διάταξή τους στο παζλ. Με την ευελιξία της, ο αλγόριθμος μπορεί να διαχειριστεί με ακρίβεια τόσο τις γραμμές όσο και τις στήλες, ανεξαρτήτως μεγέθους ή σύνθεσης του παζλ.

9.9.4 Καθορισμός Φοράς με τη `NonoDirectionExtension`

9.9.4.1 Εισαγωγή

Η απαρίθμηση `NonoDirection` και η επέκταση `NonoDirectionExtension` αποτελούν βασικά εργαλεία για τη διαχείριση των κατευθύνσεων σε σχέση με τις ενδείξεις (*clues*) ενός παζλ εικονόσταυρου. Η `NonoDirection` προσδιορίζει την κατεύθυνση πριν ή μετά από μια ένδειξη, ενώ η επέκτασή της επεκτείνει τη λειτουργικότητα για την ανάλυση δεδομένων που αφορούν την κατεύθυνση. Οι δυνατότητες αυτές είναι απαραίτητες για την ανάλυση και την επεξεργασία των στοιχείων που σχετίζονται με την τοποθέτηση ενδείξεων και τη διαχείριση των αντίστοιχων κουτιών.

9.9.4.2 Ορισμός και Τιμές της `NonoDirection`

Η `NonoDirection` είναι μια απαρίθμηση που προσδιορίζει δύο βασικές κατευθύνσεις σε σχέση με μια ένδειξη:

- `before`: Αντιπροσωπεύει την περιοχή που βρίσκεται πριν από μια ένδειξη.
- `after`: Αντιπροσωπεύει την περιοχή που βρίσκεται μετά από μια ένδειξη.

Αυτές οι κατευθύνσεις χρησιμοποιούνται σε πολλές λειτουργίες του αλγορίθμου επίλυσης για να καθορίσουν την περιοχή ενδιαφέροντος, είτε για τον εντοπισμό διαθέσιμων κουτιών είτε για την ανάλυση της τρέχουσας λύσης.

9.9.4.3 Ανάλυση Λειτουργιών που Παρέχονται από τη `NonoDirectionExtension`

9.9.4.3.1 Έλεγχος Υπαρξης Επιπλέον Ενδείξεων (`hasOtherClues`)

Η μέθοδος `hasOtherClues` καθορίζει αν υπάρχουν ενδείξεις πριν ή μετά από την τρέχουσα ένδειξη, λαμβάνοντας υπόψη την κατεύθυνση. Εξετάζει τη θέση της τρέχουσας ένδειξης (`clueIndex`) και το συνολικό μήκος της λίστας ενδείξεων (`clueListLength`).

Αν η κατεύθυνση είναι `before`, η μέθοδος επιστρέφει `true` μόνο αν ο δείκτης της ένδειξης είναι μεγαλύτερος από το μηδέν, υποδεικνύοντας την ύπαρξη προηγούμενων ενδείξεων:

```
case NonoDirection.before:
  return clueIndex > 0;
```

Στην περίπτωση της κατεύθυνσης `after`, επιστρέφει `true` μόνο αν ο δείκτης της ένδειξης είναι μικρότερος από το μήκος της λίστας ενδείξεων μείον ένα, επιβεβαιώνοντας την ύπαρξη επόμενων ενδείξεων:

```
case NonoDirection.after:
  return clueIndex < clueListLength - 1;
```

9.9.4.3.2 Επικύρωση της Λύσης (`isSolutionValid`)

Η `isSolutionValid` ελέγχει αν υπάρχουν γεμισμένα κουτιά εκτός των ορίων της πρώτης ή τελευταίας ένδειξης, ανάλογα με την κατεύθυνση.

Για την κατεύθυνση `before`, αν το `charIndex` είναι μηδέν, επιστρέφει `true`, καθώς δεν υπάρχουν προηγούμενα κουτιά. Διαφορετικά, ελέγχει το τμήμα της λύσης πριν από το `charIndex` για την ύπαρξη γεμισμένων κουτιών. Αν δεν εντοπιστούν, η λύση θεωρείται έγκυρη:

```
case NonoDirection.before:
  if (charIndex == 0) return true;
  return !solution.substring(0, charIndex - 1).split('').contains('1');
```

Στην κατεύθυνση `after`, η μέθοδος ελέγχει αν το τμήμα της λύσης μετά το τέλος του `clue` περιέχει γεμισμένα κουτιά. Αν δεν υπάρχουν, η λύση θεωρείται έγκυρη:

```
case NonoDirection.after:
  return !solution.substring(charIndex + clue + 1).split('').contains('1');
```

9.9.4.3.3 Έλεγχος Διαθέσιμων Κουτιών (`hasBoxesLeft`)

Η μέθοδος `hasBoxesLeft` ελέγχει αν υπάρχουν αρκετά διαθέσιμα κουτιά για την τοποθέτηση των υπολοίπων ενδείξεων.

Στην κατεύθυνση `before`, εξετάζει αν υπάρχουν τουλάχιστον ένα κουτί πριν από το `clue`:

```
case NonoDirection.before:
  return charIndex - 1 >= 0;
```

Για την κατεύθυνση `after`, υπολογίζει το απαιτούμενο μήκος για τις υπόλοιπες ενδείξεις και συγκρίνει αυτό το μήκος με το διαθέσιμο μήκος της λύσης:

```
case NonoDirection.after:
  return charIndex + clue + otherClues.reduce((a, b) => a + b) +
         otherClues.length - 1 < solution.length;
```

Το αποτέλεσμα καθορίζει αν η γραμμή έχει επαρκή κουτιά για να συνεχιστεί η τοποθέτηση των `clues`.

9.9.4.3.4 Απόσπαση Υπο-Συμβολοσειράς Λύσης (`getSolutionSublist`)

Η `getSolutionSublist` επιστρέφει το τμήμα της λύσης πριν ή μετά από ένα `clue`, αφαιρώντας τα κουτιά που καλύπτονται από το `clue`.

Κατεύθυνση `before`: Στην κατεύθυνση `before`, επιστρέφεται το τμήμα της λύσης από την αρχή έως το σημείο πριν από το `clue`:

```
case NonoDirection.before:
  return solution.split('').sublist(0, charIndex - 1).join();
```

Κατεύθυνση `after`: Αντίστοιχα, για την κατεύθυνση `after`, επιστρέφεται το τμήμα της λύσης μετά το `clue`:

```
case NonoDirection.after:
  return solution.split('').sublist(charIndex + clue + 1).join();
```

9.9.4.3.5 Απόσπαση Υπολίστας Ενδείξεων (`getCluesSublist`)

Η μέθοδος `getCluesSublist` επιστρέφει τις ενδείξεις πριν ή μετά από την τρέχουσα ένδειξη, ανάλογα με την κατεύθυνση.

Κατεύθυνση `before`: Στην κατεύθυνση `before`, επιστρέφεται η λίστα ενδείξεων πριν από το `clueIndex`:

```
case NonoDirection.before:
  return clues.sublist(0, clueIndex);
```

Κατεύθυνση `after`: Για την κατεύθυνση `after`, επιστρέφονται οι ενδείξεις μετά από το `clueIndex`:

```
case NonoDirection.after:
  return clues.sublist(clueIndex + 1);
```

9.9.4.3.6 Συνολική Σημασία

Οι λειτουργίες της `NonoDirectionExtension` προσφέρουν ένα ισχυρό πλαίσιο για τη διαχείριση ενδείξεων και λύσεων, ενισχύοντας την ανάλυση και την ακρίβεια του αλγορίθμου επίλυσης.

Με αυτές τις λειτουργίες, το σύστημα μπορεί:

- Να καθορίσει την εγκυρότητα της λύσης.
- Να διαχειριστεί τα διαθέσιμα κουτιά.
- Να επεξεργαστεί τα `clues` με βάση τη θέση και την κατεύθυνσή τους.

Αυτό παρέχει μεγαλύτερη ευελιξία και ακρίβεια στη διαδικασία επίλυσης, βελτιώνοντας την αποτελεσματικότητα του αλγορίθμου.

9.9.5 Επέκταση Ακεραίων με την `NonoIntExtension`

9.9.5.1 Εισαγωγή

Η `NonoIntExtension` αποτελεί μια επέκταση του τύπου δεδομένων `int` στη γλώσσα Dart, σχεδιασμένη να παρέχει μια χρήσιμη μέθοδο μορφοποίησης αριθμών. Αυτή η επέκταση χρησιμοποιείται κυρίως για να διευκολύνει την παρουσίαση μεγάλων αριθμητικών τιμών σε ένα πιο ευανάγνωστο και ευπαρουσίαστο μορφότυπο, εισάγοντας διαχωριστικά χιλιάδων.

9.9.5.2 Μέθοδος Προσθήκης Διαχωριστικών Ενδείξεων σε Αριθμούς (`dotsFormatted`)

Η κύρια μέθοδος της επέκτασης είναι η `dotsFormatted`, η οποία επιστρέφει τον ακέραιο αριθμό σε μορφοποιημένη συμβολοσειρά με διαχωριστικά χιλιάδων. Το αποτέλεσμα καθιστά πιο ευανάγνωστες τις αριθμητικές τιμές, ειδικά όταν περιέχουν πολλά ψηφία.

Η μέθοδος λειτουργεί ως εξής:

```
String get dotsFormatted => _thousandsFormatter.format(this);
```

Η μέθοδος καλείται πάνω σε έναν ακέραιο αριθμό και επιστρέφει τη μορφοποιημένη εκδοχή του χρησιμοποιώντας το NumberFormat. Για παράδειγμα, αν η μέθοδος εφαρμοστεί στον αριθμό 1000000, το αποτέλεσμα θα είναι η συμβολοσειρά "1,000,000".

9.9.5.3 Μορφή Αριθμών (_thousandsFormatter)

Το αντικείμενο _thousandsFormatter είναι υπεύθυνο για την πραγματική μορφοποίηση των αριθμών. Δημιουργείται ως μια σταθερά χρησιμοποιώντας την κλάση NumberFormat. Ο ορισμός του είναι:

```
final NumberFormat _thousandsFormatter = NumberFormat("###,###.###", "en_EN");
```

Αυτό το αντικείμενο διαμορφώνει τους αριθμούς χρησιμοποιώντας το πρότυπο "###,###.###", το οποίο εισάγει κόμμα ως διαχωριστικό χιλιάδων. Η τοπική ρύθμιση "en_EN" καθορίζει τον τρόπο μορφοποίησης σύμφωνα με τους κανόνες της αγγλικής γλώσσας.

9.9.5.4 Παράδειγμα Χρήσης

Ας εξετάσουμε πώς λειτουργεί η μέθοδος dotsFormatted μέσα από ένα παράδειγμα.

Έστω ότι θέλουμε να μορφοποιήσουμε τον αριθμό 123456789 για εμφάνιση με διαχωριστικά χιλιάδων:

```
int largeNumber = 123456789;
String formattedNumber = largeNumber.dotsFormatted;
print(formattedNumber); // "123,456,789"
```

Στο παραπάνω παράδειγμα, η μέθοδος dotsFormatted χρησιμοποιεί το _thousandsFormatter για να μετατρέψει τον αριθμό σε μια πιο ευανάγνωστη μορφή, εισάγοντας κόμματα ως διαχωριστικά χιλιάδων.

9.9.6 Επέκταση Ακεραίων με την NonoIntExtension

9.9.6.1 Εισαγωγή

Η NonoIntExtension αποτελεί μια επέκταση του τύπου δεδομένων int στη γλώσσα Dart, σχεδιασμένη να παρέχει μια χρήσιμη μέθοδο μορφοποίησης αριθμών. Αυτή η επέκταση χρησιμοποιείται κυρίως για να διευκολύνει την παρουσίαση μεγάλων αριθμητικών τιμών σε ένα πιο ευανάγνωστο και ευπαρουσίαστο μορφότυπο, εισάγοντας διαχωριστικά χιλιάδων.

9.9.6.2 Μέθοδος Προσθήκης Διαχωριστικών Ενδείξεων σε Αριθμούς (dotsFormatted)

Η κύρια μέθοδος της επέκτασης είναι η dotsFormatted, η οποία επιστρέφει τον ακέραιο αριθμό σε μορφοποιημένη συμβολοσειρά με διαχωριστικά χιλιάδων. Το αποτέλεσμα καθιστά πιο ευανάγνωστες τις αριθμητικές τιμές, ειδικά όταν περιέχουν πολλά ψηφία.

Η μέθοδος λειτουργεί ως εξής:

```
String get dotsFormatted => _thousandsFormatter.format(this);
```

Η μέθοδος καλείται πάνω σε έναν ακέραιο αριθμό και επιστρέφει τη μορφοποιημένη εκδοχή του χρησιμοποιώντας το `NumberFormat`. Για παράδειγμα, αν η μέθοδος εφαρμοστεί στον αριθμό `1000000`, το αποτέλεσμα θα είναι η συμβολοσειρά `"1,000,000"`.

9.9.6.3 Μορφή Αριθμών (`_thousandsFormatter`)

Το αντικείμενο `_thousandsFormatter` είναι υπεύθυνο για την πραγματική μορφοποίηση των αριθμών. Δημιουργείται ως μια σταθερά χρησιμοποιώντας την κλάση `NumberFormat`. Ο ορισμός του είναι:

```
final NumberFormat _thousandsFormatter = NumberFormat("###,###.###", "en_EN");
```

Αυτό το αντικείμενο διαμορφώνει τους αριθμούς χρησιμοποιώντας το πρότυπο `",."`, το οποίο εισάγει κόμμα ως διαχωριστικό χιλιάδων. Η τοπική ρύθμιση `"en_EN"` καθορίζει τον τρόπο μορφοποίησης σύμφωνα με τους κανόνες της αγγλικής γλώσσας.

9.9.6.4 Παράδειγμα Χρήσης

Ας εξετάσουμε πώς λειτουργεί η μέθοδος `dotsFormatted` μέσα από ένα παράδειγμα.

Έστω ότι θέλουμε να μορφοποιήσουμε τον αριθμό `123456789` για εμφάνιση με διαχωριστικά χιλιάδων:

```
int largeNumber = 123456789;
String formattedNumber = largeNumber.dotsFormatted;
print(formattedNumber); // "123,456,789"
```

Στο παραπάνω παράδειγμα, η μέθοδος `dotsFormatted` χρησιμοποιεί το `_thousandsFormatter` για να μετατρέψει τον αριθμό σε μια πιο ευανάγνωστη μορφή, εισάγοντας κόμματα ως διαχωριστικά χιλιάδων.

9.9.6.5 Χρήση της `NonoIntExtension` στα Παζλ Εικονόσταυρου

Η `NonoIntExtension` αξιοποιήθηκε σε διάφορα σημεία του συστήματος επίλυσης παζλ εικονόσταυρου για να παρουσιάσει δεδομένα στους χρήστες με ευανάγνωστο τρόπο. Συγκεκριμένα, εφαρμόστηκε στις τιμές που εμφανίζονται στους χρήστες, όπως:

- **Συνολικά Βήματα (Total Steps):** Ο συνολικός αριθμός βημάτων που πραγματοποιήθηκαν από τον αλγόριθμο κατά τη διαδικασία επίλυσης.
- **Γραμμές που Ελέγχθηκαν (Lines Checked):** Ο αριθμός των γραμμών που ελέγχθηκαν κατά τη διαδικασία επίλυσης, συμπεριλαμβανομένων τόσο των ολοκληρωμένων όσο και των μη ολοκληρωμένων γραμμών.
- **Κουτιά που Ελέγχθηκαν (Boxes Checked):** Ο συνολικός αριθμός κουτιών που ελέγχθηκαν και ενημερώθηκαν κατά τη διαδικασία.
- **Άλλα Κουτιά που Ελέγχθηκαν (Other Boxes Checked):** Πρόσθετα κουτιά που αναλύθηκαν κατά τη διαδικασία για την εξαγωγή πληροφοριών.
- **Συνολικά Δεδομένα Cache (Total Cache Data):** Ο αριθμός των αποθηκευμένων δεδομένων στη μνήμη cache, τα οποία χρησιμοποιήθηκαν για να μειώσουν τους επαναλαμβανόμενους υπολογισμούς.

Με τη χρήση της μεθόδου `dotsFormatted`, αυτές οι αριθμητικές τιμές εμφανίζονται με διαχωριστικά χιλιάδων, διευκολύνοντας την κατανόηση και οπτικοποίησή τους από τους χρήστες. Για παράδειγμα, ένας αριθμός όπως `10000` εμφανίζεται ως `"10,000"`, καθιστώντας τον πιο ευανάγνωστο και φιλικό προς τον χρήστη.

Η εφαρμογή αυτής της μορφοποίησης εξασφαλίζει ότι οι χρήστες μπορούν να κατανοούν άμεσα τη σημασία των αριθμητικών δεδομένων, ενισχύοντας την εμπειρία χρήσης και παρέχοντας μια πιο επαγγελματική εμφάνιση των πληροφοριών.

9.9.6.6 Συμπέρασμα

Η `NonoIntExtension` είναι μια χρήσιμη και απλή επέκταση που ενισχύει τη διαχείριση ακέραιων αριθμών στα πλαίσια της ανάπτυξης λογισμικού για παζλ εικονόσταυρου. Με τη μορφοποίηση αριθμών μέσω της μεθόδου `dotsFormatted`, προσφέρει μεγαλύτερη ευκρίνεια και αναγνωσιμότητα στα αποτελέσματα και τα δεδομένα που εμφανίζονται.

9.9.7 Επέκταση Λιστών (NonoListExtension)

9.9.7.1 Εισαγωγή

Η `NonoListExtension` παρέχει επιπλέον λειτουργίες για λίστες που χρησιμοποιούνται κατά τη διαδικασία επίλυσης ενός παζλ εικονόσταυρου. Αυτή η επέκταση περιλαμβάνει βοηθητικές μεθόδους που εφαρμόζονται σε διαφορετικούς τύπους λιστών, βελτιώνοντας τη λειτουργικότητά τους και διευκολύνοντας τις πράξεις στον αλγόριθμο επίλυσης.

9.9.7.2 Επέκταση Λιστών Συμβολοσειρών (NonoStringListExtension)

Η `NonoStringListExtension` στοχεύει στη διευκόλυνση της διαχείρισης λιστών χαρακτήρων που σχετίζονται με την τρέχουσα κατάσταση της λύσης. Παρέχει εργαλεία για τον άμεσο έλεγχο της κατάστασης των στοιχείων μιας λίστας.

9.9.7.2.1 Έλεγχος αν το κάθε στοιχείο είναι μηδέν (everyElementIsZero)

Η μέθοδος `everyElementIsZero` χρησιμοποιείται για να ελέγξει αν όλα τα στοιχεία μιας λίστας χαρακτήρων είναι 0. Ο κώδικας είναι:

```
bool get everyElementIsZero => every((String e) => e == '0');
```

Η μέθοδος διατρέχει κάθε στοιχείο της λίστας και επιστρέφει `true` μόνο αν όλα τα στοιχεία ισούνται με 0. Είναι χρήσιμη για την επιβεβαίωση ότι μια γραμμή ή στήλη αποτελείται αποκλειστικά από διαγραμμένα κουτιά.

9.9.7.3 Επέκταση Λιστών Ακεραίων (NonoIntListExtension)

Η `NonoIntListExtension` επεκτείνει τις δυνατότητες των λιστών ακέραιων αριθμών που χρησιμοποιούνται για ενδείξεις (`clues`) και υπολογισμούς που σχετίζονται με αυτές. Παρέχει λειτουργίες για την εύρεση αθροισμάτων, ελάχιστων και μέγιστων σημείων εκκίνησης.

9.9.7.3.1 Άθροισμα Στοιχείων (sum)

Η μέθοδος sum υπολογίζει το συνολικό άθροισμα των στοιχείων μιας λίστας:

```
int get sum => fold<int>(0, (int previousValue, int element) => previousValue + element);
```

Η μέθοδος προσθέτει κάθε στοιχείο στο τρέχον άθροισμα. Είναι χρήσιμη για τον υπολογισμό του συνολικού μήκους των ενδείξεων μιας γραμμής ή στήλης.

9.9.7.3.2 Μικρότερο Ελάχιστο Σημείο Εκκίνησης (minStartingPoint)

Η `minStartingPoint` υπολογίζει το ελάχιστο σημείο εκκίνησης για μια ένδειξη με βάση τα προηγούμενα στοιχεία:

```
int minStartingPoint(int index) => index == 0 ? 0
  : take(index).reduce((int value, int element) => value + element + 1);
```

Η μέθοδος λαμβάνει υπόψη τα προηγούμενα στοιχεία και τα κενά μεταξύ τους για να υπολογίσει την ελάχιστη θέση εκκίνησης.

9.9.7.3.3 Μέγιστο Σημείο Εκκίνησης (maxStartingPoint)

Η `maxStartingPoint` υπολογίζει τη μέγιστη δυνατή θέση εκκίνησης μιας ένδειξης:

```
int maxStartingPoint(int index, int totalLength) =>
  index == length - 1 ? totalLength
  : totalLength - sublist(index + 1).reduce((int value, int element) => value + element + 1)
  - this[index];
```

Η μέθοδος λαμβάνει υπόψη τα στοιχεία που ακολουθούν και το συνολικό μήκος της γραμμής ή στήλης για να προσδιορίσει τη μέγιστη θέση εκκίνησης.

9.9.7.4 Επέκταση Λιστών Χαρτών (NonoMapListExtension)

Η `NonoMapListExtension` είναι σχεδιασμένη για τη διαχείριση λιστών που περιέχουν χάρτες με δείκτες και άξονες. Αυτή η επέκταση διευκολύνει τη λειτουργία του σωρού (*stack*) που χρησιμοποιείται για την αποθήκευση γραμμών και στηλών προς επεξεργασία.

9.9.7.4.1 Έλεγχος Ύπαρξης σε Στοίβα (`isInStack`)

Η μέθοδος `isInStack` επαληθεύει αν ένας συγκεκριμένος δείκτης και τύπος άξονα περιέχεται στον σωρό:

```
bool isInStack(int charIndex, NonoAxis lineType) {
  return contains(<int, NonoAxis>{charIndex: lineType});
}
```

Αυτή η λειτουργία βοηθά στην αποφυγή διπλών εγγραφών, εξασφαλίζοντας ότι κάθε στοιχείο στον σωρό είναι μοναδικό.

9.9.7.4.2 Εύρεση Καινούργιων Στοιχείων Στοίβας (getNewStackElements)

9.9.7.4.2.1 Εισαγωγή

Η μέθοδος getNewStackElements είναι υπεύθυνη για τη δημιουργία νέων στοιχείων του σωρού (*stack*) που περιλαμβάνουν τις επηρεασμένες γραμμές ή στήλες σε ένα παζλ εικονόσταυρου. Αυτή η διαδικασία είναι κρίσιμη για την ενημέρωση του σωρού με νέες γραμμές ή στήλες που επηρεάζονται από πρόσφατες αλλαγές:

```
List<Map<int, NonoAxis>> getNewStackElements(List<int> charIndexes, NonoAxis lineType) {
  final NonoAxis newAxis = lineType == NonoAxis.row ? NonoAxis.column : NonoAxis.row;
  return charIndexes.map((int charIndex) => <int, NonoAxis>{charIndex: newAxis}).toList();
}
```

Η μέθοδος προσθέτει γραμμές ή στήλες στον σωρό, με τον νέο άξονα να είναι ο αντίθετος από αυτόν που δόθηκε.

9.9.7.4.2.2 Εφαρμογή

Το πρώτο βήμα στη λειτουργία της μεθόδου είναι να καθορίσει τον νέο άξονα για τα στοιχεία που θα προστεθούν. Αν ο άξονας της γραμμής που επεξεργαζόμαστε είναι γραμμή (NonoAxis.row), τότε ο νέος άξονας για τα επηρεασμένα στοιχεία είναι στήλη (NonoAxis.column) και αντίστροφα. Αυτή η εναλλαγή εξασφαλίζει ότι οι αλλαγές σε μια γραμμή επηρεάζουν τις αντίστοιχες στήλες και το αντίθετο.

Έπειτα, η μέθοδος διατρέχει τις θέσεις των χαρακτήρων (charIndexes) που παρέχονται ως είσοδος και δημιουργεί έναν χάρτη (Map) για κάθε θέση, αντιστοιχίζοντας τη θέση με τον νέο άξονα (NonoAxis). Αυτό επιτυγχάνεται μέσω της χρήσης της μεθόδου map, η οποία εφαρμόζει μια συνάρτηση σε κάθε στοιχείο της λίστας και επιστρέφει μια νέα λίστα.

Για παράδειγμα, εάν οι θέσεις των χαρακτήρων είναι [0, 1, 2] και ο τρέχων άξονας είναι NonoAxis.row, η μέθοδος θα δημιουργήσει τους χάρτες:

```
[ {0: NonoAxis.column}, {1: NonoAxis.column}, {2: NonoAxis.column} ]
```

Αυτή η λίστα χαρτών επιστρέφεται από τη μέθοδο, ώστε να χρησιμοποιηθεί για την ενημέρωση του σωρού.

9.9.7.4.2.3 Λειτουργική Σημασία

Η λειτουργία της μεθόδου getNewStackElements διασφαλίζει ότι όλες οι επηρεασμένες γραμμές ή στήλες λαμβάνονται υπόψη, προσφέροντας σαφή και αποδοτική ενημέρωση του σωρού. Με αυτή τη διαδικασία, το σύστημα είναι σε θέση να διαχειρίζεται αλλαγές με ακρίβεια και να επεξεργάζεται τα δεδομένα του παζλ εικονόσταυρου αποτελεσματικά.

9.9.7.4.3 Ανανέωση Στοίβας (updateStack)

9.9.7.4.3.1 Εισαγωγή

Η μέθοδος updateStack ενημερώνει τον σωρό με νέα στοιχεία, επανατοποθετώντας διπλότυπα. Ο κώδικας της μεθόδου είναι ο εξής:

```
List<Map<int, NonoAxis>> updateStack(List<Map<int, NonoAxis>> newStackElements) {
  for (final Map<int, NonoAxis> element in newStackElements) {
```

```
    final int duplicateIndex = indexWhere((Map<int, NonoAxis> e) =>
        e.keys.first == element.keys.first && e.values.first == element.values.first);
    if (duplicateIndex >= 0) {
      remove(element);
      insert((duplicateIndex / 2).ceil(), element);
    } else {
      add(element);
    }
  }
  return this;
}
```

Η μέθοδος διαχειρίζεται τη σειρά των στοιχείων στον σωρό, διασφαλίζοντας ότι αυτά που έχουν σημασία βρίσκονται σε προτεραιότητα.

9.9.7.4.3.2 Εφαρμογή

Η `updateStack` διατρέχει τη λίστα με τα νέα στοιχεία του σωρού (`newStackElements`) και ελέγχει για κάθε στοιχείο αν υπάρχει ήδη στον σωρό. Για τον έλεγχο αυτό, χρησιμοποιεί τη μέθοδο `indexWhere`, η οποία εντοπίζει τη θέση του στοιχείου που ταιριάζει με τα δεδομένα του νέου στοιχείου.

Εάν το στοιχείο υπάρχει ήδη, αφαιρείται από τη λίστα και εισάγεται εκ νέου στη μέση του σωρού. Διαφορετικά, προστίθεται στο τέλος της λίστας. Η λογική της μεθόδου μπορεί να συνοψιστεί στον παρακάτω ψευδοκώδικα:

- Για κάθε νέο στοιχείο:
  - Ελέγχει αν το στοιχείο υπάρχει στον σωρό.
  - Αν υπάρχει, αφαιρείται και εισάγεται στη μέση του σωρού.
  - Αν δεν υπάρχει, προστίθεται στο τέλος της λίστας.

Για παράδειγμα, αν ο αρχικός σωρός είναι:

`[0: NonoAxis.row, 1: NonoAxis.column]`

και τα νέα στοιχεία είναι:

`[1: NonoAxis.column, 2: NonoAxis.row]`

τότε ο ενημερωμένος σωρός θα είναι:

`[0: NonoAxis.row, 2: NonoAxis.row, 1: NonoAxis.column]`

9.9.7.4.3.3 Αποτίμηση της Μεθόδου

Παρά την αρχική πρόθεση βελτίωσης της συνολικής απόδοσης, η μέθοδος `updateStack` δεν ανταποκρίθηκε στις προσδοκίες. Ο χρόνος που απαιτούνταν για:

- τον εντοπισμό των στοιχείων,

- την αφαίρεσή τους από την τρέχουσα θέση στη λίστα,
- την εισαγωγή τους στη μέση ή στο τέλος του σωρού,

σε συνδυασμό με την τυχαιότητα της συμβολής τους στη λύση, προκάλεσε σημαντική επιβάρυνση στην απόδοση του αλγορίθμου.

Οι επαναλαμβανόμενες ενέργειες αφαίρεσης και εισαγωγής στοιχείων αύξησαν την πολυπλοκότητα, χωρίς να εξασφαλίσουν ουσιαστική βελτίωση. Ως αποτέλεσμα, η μέθοδος δεν κρίθηκε κατάλληλη για χρήση στο τελικό σύστημα.

9.9.7.5 Συμπέρασμα

Η `NonoListExtension` αποτελεί ένα σύνολο εργαλείων που βελτιώνουν τη διαχείριση των λιστών στο πλαίσιο της επίλυσης παζλ εικονόσταυρου. Με τις τρεις διακριτές επεκτάσεις της, απλοποιεί την εργασία με χαρακτήρες, αριθμούς και χάρτες, ενώ παρέχει ισχυρές δυνατότητες που ενισχύουν τη συνολική αποτελεσματικότητα και σαφήνεια του κώδικα.

9.9.8 Επέκταση Συμβολοσειρών (`NonoStringExtension`)

9.9.8.1 Εισαγωγή

Η `NonoStringExtension` αποτελεί μια επέκταση της κλάσης String που παρέχει εξειδικευμένες λειτουργίες για τη διαχείριση συμβολοσειρών στα παζλ εικονόσταυρου. Μέσω αυτών των μεθόδων, βελτιώνεται η δυνατότητα επεξεργασίας και ανάλυσης δεδομένων που σχετίζονται με τις συμβολοσειρές λύσης, όπως η ανάκτηση χαρακτήρων, η απομόνωση γραμμών ή στηλών, και η επαλήθευση στοιχείων.

Η επέκταση προσθέτει νέες μεθόδους στη βασική κλάση String, οι οποίες είναι ειδικά σχεδιασμένες για τη λειτουργία του αλγορίθμου επίλυσης των παζλ εικονόσταυρου. Κάθε μία από αυτές τις μεθόδους έχει ως στόχο να επιλύσει συγκεκριμένα προβλήματα που σχετίζονται με την επεξεργασία των δεδομένων του παζλ, όπως τον υπολογισμό του αθροίσματος των γεμισμένων κουτιών ή την απομόνωση συγκεκριμένων γραμμών και στηλών.

9.9.8.2 Ανάκτηση Χαρακτήρα από Σημείο στη Συμβολοσειρά (`characterAt`)

Η μέθοδος `characterAt` εξυπηρετεί την άμεση ανάκτηση ενός χαρακτήρα από μια συμβολοσειρά, με βάση τον δείκτη του (`index`). Αυτή η λειτουργία είναι θεμελιώδης όταν χρειάζεται να αναλυθεί η λύση του παζλ ανά χαρακτήρα, διευκολύνοντας τη διαχείριση των κουτιών της γραμμής ή στήλης.

Η μέθοδος λειτουργεί ως εξής: Η συμβολοσειρά χωρίζεται σε μια λίστα χαρακτήρων μέσω της `split('')`. Έπειτα, επιστρέφεται το στοιχείο που βρίσκεται στη θέση που αντιστοιχεί στο παρεχόμενο `index`. Αυτή η προσέγγιση εξασφαλίζει άμεση πρόσβαση στον επιθυμητό χαρακτήρα.

```
String characterAt(int index) => split('')[index];
```

Παράδειγμα: Για τη συμβολοσειρά `"12345"`, αν καλέσουμε `characterAt(2)`, η μέθοδος επιστρέφει `"3"`. Η συμβολοσειρά μετατρέπεται σε λίστα `['1', '2', '3', '4', '5']` και ο αλγόριθμος επιλέγει το στοιχείο στη θέση 2.

9.9.8.3 Εύρεση Συνόλου Συμπληρωμένων Στοιχείων (`sumFilledBoxes`)

Η `sumFilledBoxes` υπολογίζει το άθροισμα όλων των γεμισμένων κουτιών σε μια συμβολοσειρά. Αυτή η μέθοδος είναι ιδιαίτερα χρήσιμη για την ανάλυση της προόδου στην επίλυση μιας γραμμής ή στήλης, παρακάμπτοντας τα κενά κουτιά (`?`).

Η λειτουργία χρησιμοποιεί τη μέθοδο `split('')` για να διασπάσει τη συμβολοσειρά σε χαρακτήρες. Στη συνέχεια, εφαρμόζει την `fold`, η οποία διατρέχει κάθε χαρακτήρα και ελέγχει αν είναι διαφορετικός από `?`. Εάν ναι, ο χαρακτήρας μετατρέπεται σε ακέραιο με την `int.parse` και προστίθεται στο συνολικό άθροισμα.

```
int get sumFilledBoxes =>
    split('').fold(0, (int prev, String element) =>
        prev + (element != '?' ? int.parse(element) : 0));
```

Παράδειγμα: Για τη συμβολοσειρά `"1?30"`, η μέθοδος αγνοεί το `?` και υπολογίζει το άθροισμα: `1 + 3 + 0 = 4`.

9.9.8.4 Σύγκριση Στοιχείων (`isSameClueIndexWith`)

Η μέθοδος `isSameClueIndexWith` συγκρίνει δύο χαρακτήρες για να διαπιστώσει αν ανήκουν στο ίδιο *index* ενδείξεων. Αυτή η λειτουργία χρησιμοποιείται για την επαλήθευση της ακρίβειας στη διάταξη των ενδείξεων και των γεμισμένων κουτιών.

Ο αλγόριθμος ελέγχει αν οι δύο χαρακτήρες είναι ίσοι και ταυτόχρονα διαφορετικοί από τους χαρακτήρες `?` και `0`. Εάν πληρούνται αυτές οι συνθήκες, η μέθοδος επιστρέφει `true`.

```
bool isSameClueIndexWith(String element) =>
    this == element && this != '?' && element != '0';
```

Παράδειγμα: Για τη σύγκριση των χαρακτήρων `"3"` και `"3"`, η μέθοδος επιστρέφει `true`. Αν συγκρίνουμε `"?"` με `"3"`, επιστρέφει `false`.

9.9.8.5 Εύρεση και Επιστροφή Γραμμής (`getRowIsolate`)

Η `getRowIsolate` απομονώνει μια γραμμή από τη συνολική λύση του παζλ, με βάση τον δείκτη της γραμμής (`lineIndex`) και το πλάτος της (`width`). Αυτή η λειτουργία είναι απαραίτητη για την ανάλυση συγκεκριμένων γραμμών.

Ο αλγόριθμος υπολογίζει το εύρος των χαρακτήρων που αντιστοιχούν στη γραμμή, χρησιμοποιώντας τα όρια `lineIndex * width` και `(lineIndex + 1) * width`. Έπειτα, εξάγει τους χαρακτήρες με τη `getRange`, τους επανασυνθέτει με την `join`, και αφαιρεί ανεπιθύμητους χαρακτήρες με την `replaceAll`.

```
String getRowIsolate(int lineIndex, int width) =>
    split('').getRange(lineIndex * width, width * (lineIndex + 1))
             .join()
             .replaceAll(RegExp(r'[ (),]'), '');
```

Παράδειγμα: Για τη συμβολοσειρά `"123456789"`, αν καλέσουμε `getRowIsolate(0, 3)`, η μέθοδος επιστρέφει `"123"`.

9.9.8.6 Εύρεση και Επιστροφή Στήλης (`getColumnIsolate`)

Η `getColumnIsolate` απομονώνει μια στήλη από τη συνολική λύση, χρησιμοποιώντας τον δείκτη της γραμμής (`lineIndex`) και το πλάτος του παζλ (`width`). Η μέθοδος συλλέγει τους χαρακτήρες της στήλης μέσω επαναληπτικής διαδικασίας.

Ο αλγόριθμος ξεκινά από τη θέση `lineIndex` και, αυξάνοντας κατά `width` σε κάθε βήμα, προσθέτει τους χαρακτήρες της στήλης στη μεταβλητή `columnSol`. Τέλος, επιστρέφει τη στήλη ως συμβολοσειρά.

```
String getColumnIsolate(int lineIndex, int width) {
  String columnSol = '';
  for (int solChar = lineIndex; solChar < length; solChar += width) {
    columnSol += this[solChar];
  }
  return columnSol;
}
```

Παράδειγμα: Για τη συμβολοσειρά `"123456789"`, αν καλέσουμε `getColumnIsolate(0, 3)`, η μέθοδος επιστρέφει `"147"`.

Κεφάλαιο 10ο: Σχεδιασμός και Προσθήκης Παζλ

10.1 Εισαγωγή

Η ενότητα αυτή επικεντρώνεται στη διαδικασία σχεδιασμού, διαμόρφωσης και επεξεργασίας παζλ τύπου Nonogram. Η σελίδα δημιουργίας Nonogram σχεδιάστηκε με γνώμονα τόσο τη λειτουργικότητα όσο και την εμπειρία του χρήστη, επιτρέποντας την εύκολη δημιουργία και τροποποίηση πλέγματος μέσω διαδραστικών εργαλείων. Τα χαρακτηριστικά της σελίδας περιλαμβάνουν τη δυνατότητα προσθήκης ενδείξεων, τη δυναμική προσαρμογή του μεγέθους του πλέγματος και την επεξεργασία της κατάστασης κάθε κουτιού μέσω κινήσεων (gestures).

Το κεφάλαιο αναλύει διεξοδικά τις βασικές λειτουργίες που παρέχονται στον χρήστη, όπως η προσθήκη ενδείξεων, η σχεδίαση του παζλ μέσω ζωγραφικής και η προσαρμογή του μεγέθους του πλέγματος. Κάθε μία από αυτές τις λειτουργίες συνοδεύεται από τεχνικές λεπτομέρειες σχετικά με την υλοποίησή τους, αναδεικνύοντας την πολυπλοκότητα των αλγορίθμων και τον τρόπο που αυτές διασφαλίζουν την απόδοση και την ακρίβεια του συστήματος.

Η ευελιξία που προσφέρει η δυνατότητα αλλαγής διαστάσεων (ύψους και πλάτους) του πλέγματος, καθώς και η ταυτόχρονη ανανέωση των στοιχείων και της λύσης του παζλ, αποτελεί ένα από τα πιο ισχυρά χαρακτηριστικά του εργαλείου. Η ενσωμάτωση χαρακτηριστικών όπως η αυτόματη ενημέρωση των ενδείξεων και η προσαρμογή τους στις αλλαγές του πλέγματος ενισχύουν την αποτελεσματικότητα της δημιουργικής διαδικασίας.

Στο σύνολό της, η σελίδα δημιουργίας Nonogram αντιπροσωπεύει μια ισορροπία μεταξύ της τεχνικής πολυπλοκότητας και της απλότητας στη χρήση. Το περιβάλλον χρήστη έχει σχεδιαστεί έτσι ώστε να παρέχει μια διαισθητική και φιλική εμπειρία, ενώ παράλληλα η υποκείμενη υποδομή της εφαρμογής διαχειρίζεται την πολυπλοκότητα με τρόπο που να μην επηρεάζει την ομαλή λειτουργία. Μέσα από τα επόμενα υποκεφάλαια, αναλύεται διεξοδικά κάθε πτυχή της σελίδας δημιουργίας, προσφέροντας μια ολοκληρωμένη εικόνα της σχεδίασης και της λειτουργίας της.

10.2 Ευρύτερη Δομή και Ροή Ενότητας

Η ενότητα αυτή περιγράφει τη δομή και τη λειτουργικότητα της σελίδας δημιουργίας Nonogram, αναλύοντας τα βασικά στοιχεία και τις διαδικασίες που τη συνθέτουν. Στόχος της είναι να παρέχει μια συνολική εικόνα του τρόπου λειτουργίας της σελίδας, ξεκινώντας από τα χαρακτηριστικά και τη γενική ροή εργασίας, και καταλήγοντας στις λεπτομέρειες των επιμέρους λειτουργιών.

Η δομή της ενότητας ακολουθεί μια φυσική ροή που ξεκινά με την περιγραφή των βασικών χαρακτηριστικών της εφαρμογής, όπως η δυνατότητα προσαρμογής του μεγέθους του πλέγματος, η επεξεργασία κουτιών και η δυναμική ενημέρωση των ενδείξεων. Κάθε λειτουργικότητα παρουσιάζεται με τρόπο που να ενσωματώνει τόσο τις τεχνικές λεπτομέρειες της υλοποίησης όσο και την επίδρασή της στη συνολική εμπειρία χρήστη.

10.2.1 Δομή Ενότητας

Η ενότητα χωρίζεται σε υποενότητες που καλύπτουν τα εξής θέματα:

- **Χαρακτηριστικά της Σελίδας Δημιουργίας:** Παρουσιάζονται τα βασικά εργαλεία και οι επιλογές που παρέχονται στον χρήστη, όπως οι κινήσεις (gestures) για την ενημέρωση της κατάστασης των κουτιών, οι δυνατότητες προσαρμογής του μεγέθους του πλέγματος, και η λειτουργία επίλυσης παζλ.
- **Δημιουργία Παζλ μέσω Στοιχείων και Ζωγραφικής:** Εξηγείται ο τρόπος που ο χρήστης μπορεί να σχεδιάσει ένα παζλ εισάγοντας ενδείξεις ή χρησιμοποιώντας τη λειτουργία ζωγραφικής για την τροποποίηση της λύσης.

- **Προσαρμογή Διαστάσεων Πλέγματος:** Παρουσιάζεται η δυνατότητα προσαρμογής του ύψους και του πλάτους του πλέγματος, με ταυτόχρονη ανανέωση των στοιχείων και της λύσης.
- **Τεχνική Υλοποίηση:** Παρουσιάζονται τα υποκείμενα τεχνικά εργαλεία και οι αλγόριθμοι που υποστηρίζουν τη λειτουργικότητα της σελίδας. Εξηγείται η δομή των δεδομένων, οι μέθοδοι που χρησιμοποιούνται για την ενημέρωση του πλέγματος, καθώς και οι τεχνικές διαχείρισης της λύσης.
- **Εμπειρία Χρήστη και Λειτουργικότητα:** Εξετάζεται ο τρόπος με τον οποίο η σχεδίαση της σελίδας συμβάλλει στην ευχρηστία της εφαρμογής. Η ροή εργασίας είναι σχεδιασμένη έτσι ώστε να παρέχει άμεση ανατροφοδότηση στον χρήστη, επιτρέποντας τη γρήγορη και εύκολη προσαρμογή του πλέγματος και των ενδείξεων.

Στο σύνολό της, η ενότητα αυτή εστιάζει στο να εξηγήσει πώς η σελίδα δημιουργίας Nonogram καταφέρνει να ισορροπήσει ανάμεσα στην τεχνική αρτιότητα και τη φιλικότητα προς τον χρήστη. Καθώς προχωράει η ανάλυση, οι αναγνώστες αποκτούν μια βαθύτερη κατανόηση των μηχανισμών και της αρχιτεκτονικής που υποστηρίζουν τη δημιουργία και την επεξεργασία ενός Nonogram, ενώ ταυτόχρονα αναγνωρίζουν τη σημασία της λειτουργικότητας αυτής στη συνολική εμπειρία χρήστη της εφαρμογής.

10.3 Χαρακτηριστικά

10.3.1 Εισαγωγή

Η σελίδα δημιουργίας **Εικονόσταυρου (Nonogram)** διαθέτει μια ποικιλία χαρακτηριστικών που έχουν σχεδιαστεί για να προσφέρουν μια **ευέλικτη**, **αποδοτική** και **φιλική προς τον χρήστη εμπειρία**. Ο σχεδιασμός της επιτρέπει στους χρήστες να δημιουργούν, να προσαρμόζουν και να επεξεργάζονται παζλ με τρόπο που να ανταποκρίνεται στις προσωπικές τους ανάγκες και προτιμήσεις, ενώ υποστηρίζει τη δυναμική ανατροφοδότηση σε πραγματικό χρόνο. Παρακάτω περιγράφονται τα βασικά χαρακτηριστικά της.

10.3.2 Δημιουργία Παζλ με Εισαγωγή Στοιχείων

Οι χρήστες μπορούν να εισάγουν τις **ενδείξεις** (clues) απευθείας μέσω ενός πεδίου κειμένου. Αυτή η λειτουργία υποστηρίζεται από το `TextEditingController`, το οποίο διαχειρίζεται τις αλλαγές που γίνονται στο πεδίο κειμένου. Οι αλλαγές στις ενδείξεις ενημερώνονται δυναμικά στο πλέγμα μέσω του `CreateNonogramCubit`, διασφαλίζοντας ότι κάθε τροποποίηση έχει **άμεση οπτική ανατροφοδότηση**.

10.3.3 Ζωγραφική στο Πλέγμα

Η δυνατότητα ζωγραφικής επιτρέπει στους χρήστες να αλληλεπιδρούν απευθείας με το πλέγμα χρησιμοποιώντας **gestures**, όπως taps και pan gestures. Οι χρήστες μπορούν να γεμίσουν ή να διαγράψουν κουτιά, και οι αλλαγές τους αντικατοπτρίζονται στη λύση του παζλ. Αυτό επιτυγχάνεται μέσω του `NonogramGrid` και του `GridGestures`, τα οποία αναλαμβάνουν τον χειρισμό των gestures και την ενημέρωση της κατάστασης.

10.3.4 Προσαρμογή Διαστάσεων Πλέγματος

Οι χρήστες μπορούν να αλλάξουν το μέγεθος του πλέγματος μέσω **sliders** που παρέχονται για το πλάτος και το ύψος. Κάθε αλλαγή στο μέγεθος οδηγεί σε **δυναμική ενημέρωση** τόσο του πλέγματος όσο και των ενδείξεων, διασφαλίζοντας τη συνέπεια και την ακρίβεια στη διαχείριση της λύσης.

10.3.5 Ανατροφοδότηση σε Πραγματικό Χρόνο

Κάθε αλλαγή που γίνεται στο πλέγμα, είτε μέσω της προσθήκης ή αφαίρεσης στοιχείων είτε μέσω της προσαρμογής των διαστάσεων, αντικατοπτρίζεται άμεσα στη λύση και στις ενδείξεις. Η συνεχής ανατροφοδότηση επιτρέπει στους χρήστες να βλέπουν **άμεσα** πώς επηρεάζονται τα δεδομένα του παζλ, ενισχύοντας την κατανόηση και την ακρίβεια.

10.3.6 Δυνατότητα Επίλυσης του Παζλ

Η σελίδα περιλαμβάνει κουμπί `Solve`, το οποίο επιτρέπει στους χρήστες να μεταβούν απευθείας στη σελίδα επίλυσης του παζλ. Αυτή η δυνατότητα διευκολύνει τη **γρήγορη δοκιμή** και αξιολόγηση της λύσης που έχει δημιουργηθεί.

10.3.7 Προσαρμοσμένες Ρυθμίσεις Επεξεργασίας

Παρέχεται η επιλογή προσαρμογής του τρόπου που επηρεάζουν οι **κινήσεις (gestures)** αφής το πλέγμα. Οι χρήστες μπορούν να επιλέξουν αν οι αλλαγές θα εφαρμόζονται κατά τη διάρκεια της κίνησης ή μόνο στο τέλος αυτής, προσφέροντας μεγαλύτερη **ακρίβεια** στην επεξεργασία.

10.3.8 Φιλικός προς τον Χρήστη Σχεδιασμός

Η διεπαφή είναι σχεδιασμένη ώστε να είναι **απλή** και **διαισθητική**, με ξεκάθαρα στοιχεία χειρισμού. Ο σχεδιασμός της εξασφαλίζει ότι τόσο οι αρχάριοι όσο και οι έμπειροι χρήστες μπορούν να επωφεληθούν από τα εργαλεία που παρέχονται, χωρίς να χρειάζεται να αφιερώσουν χρόνο για εκμάθηση.

10.3.9 Διαχείριση Δεδομένων

Η λύση του παζλ και οι ενδείξεις αποθηκεύονται σε μορφή **string**, η οποία επιτρέπει την εύκολη διαχείριση, ανάλυση και επεξεργασία τους από τον αλγόριθμο. Οι ενδείξεις μπορούν να ενημερώνονται **δυναμικά** με βάση τις αλλαγές που γίνονται στο πλέγμα, διασφαλίζοντας τη συνέπεια μεταξύ λύσης και ενδείξεων.

10.4 Δημιουργία Παζλ μέσω Στοιχείων

10.4.1 Εισαγωγή

Η λειτουργία **εισαγωγής στοιχείων** επιτρέπει στους χρήστες να σχεδιάσουν το παζλ τους με βάση τις **ενδείξεις** για κάθε γραμμή και στήλη. Αυτή η ενότητα περιγράφει πώς υλοποιείται αυτή η δυνατότητα, τόσο από την άποψη της **εμπειρίας χρήστη** όσο και της **τεχνικής υποδομής**.

10.4.2 Εισαγωγή Ενδείξεων

Οι **ενδείξεις** εισάγονται μέσω ενός **πεδίου κειμένου** που χειρίζεται το `TextEditingController`. Οι χρήστες μπορούν να προσθέσουν ή να τροποποιήσουν τις ενδείξεις εισάγοντας **αριθμούς διαχωρισμένους με κόμματα** (π.χ., ”1, 2, 3”). Όταν η εισαγωγή αλλάζει, το `CreateNonogramCubit` ενημερώνει τη σχετική **γραμμή** ή **στήλη**, διατηρώντας τη **συνοχή** μεταξύ των **ενδείξεων** και της **λύσης**.

```
K      10
```

```
TextFormField(
  controller: textEditingController,
  onChanged: createNonogramCubit.updateSelectedLine,
)
```

10.4.3 Διαχείριση Επιλεγμένης Γραμμής ή Στήλης

Οι χρήστες μπορούν να **επιλέξουν** τη γραμμή ή τη στήλη που θέλουν να τροποποιήσουν πατώντας επάνω της στο **πλέγμα**. Η μέθοδος `setSelectedLine` στο `CreateNonogramCubit` καθορίζει τη γραμμή ή τη στήλη που επεξεργάζεται ο χρήστης.

```
createNonogramCubit.setSelectedLine(axis, index, cluesLine);
```

Η κατάσταση της **επιλεγμένης γραμμής** αποθηκεύεται στο `SelectedLine`, μια **κλάση** που περιλαμβάνει τις παρακάτω πληροφορίες:

- **Άξονας (Axis):** Οριζόντιος ή κάθετος.
- **Δείκτης (Index):** Η θέση της γραμμής ή στήλης.
- **Ενδείξεις (Clues):** Η λίστα των αριθμών που αντιπροσωπεύουν τη γραμμή ή τη στήλη.

10.5 Δημιουργία Παζλ μέσω Ζωγραφικής

10.5.1 Εισαγωγή

Η μέθοδος `updateBox` του `CreateNonogramCubit` διαχειρίζεται την ενημέρωση ενός κουτιού. Όταν ένα κουτί πατηθεί ή σύρεται, η κατάστασή του αλλάζει, και οι αλλαγές αντικατοπτρίζονται τόσο στη λύση όσο και στις ενδείξεις.

10.5.2 Αλληλεπίδραση με το Πλέγμα

Η ζωγραφική στο πλέγμα υλοποιείται μέσω του `NonogramGrid`, ενός *stateful widget* που υποστηρίζει κινήσεις όπως *taps* και *pan gestures*. Οι χρήστες μπορούν να:

- Πατήσουν σε ένα κουτί για να το γεμίσουν ή να το διαγράψουν.
- Σύρουν το δάχτυλό τους πάνω από κουτιά για μαζικές αλλαγές.

Τα *gestures* διαχειρίζονται μέσω του `GridGestures`, το οποίο περιλαμβάνει τρεις κύριες μεθόδους:

- **`onTap`**: Ενημερώνει την κατάσταση ενός κουτιού όταν πατηθεί.
- **`onPan`**: Ενημερώνει την κατάσταση πολλαπλών κουτιών καθώς ο χρήστης σύρει το δάχτυλό του.
- **`onPanEnd`**: Εκτελείται στο τέλος μιας σύρσης για να εξασφαλίσει την ολοκλήρωση των αλλαγών.

```
GridGestures(
  onTap: createNonogramCubit.updateBox,
  onPan: createNonogramCubit.onPan,
  onPanEnd: state.editingSettings.updateOnPanEnd ? createNonogramCubit.onPanEnd : null,
)
```

10.5.3 Διαχείριση Καταστάσεων Κουτιών

Η ενημέρωση της κατάστασης ενός κουτιού βασίζεται στη μέθοδο `updateBox` του `CreateNonogramCubit`. Όταν ένα κουτί γεμίζεται, αλλάζει από την κατάσταση `'?'` (άδειο) στην `'1'` (γεμάτο). Αν διαγραφεί, επιστρέφει στην κατάσταση `'?'`.

```
final bool isEmpty =
    state.solution.characterAt(boxIndex) == '?';

emit(state.copyWith(
    solution: state.solution.replaceRange(
        boxIndex,
        boxIndex + 1,
        isEmpty ? '1' : '?'
    )
));
```

10.5.4 Συγχρονισμός Ζωγραφικής και Ενδείξεων

Μετά από κάθε αλλαγή στην κατάσταση ενός κουτιού, ενημερώνονται αυτόματα οι ενδείξεις (*clues*) της αντίστοιχης γραμμής ή στήλης. Αυτό γίνεται μέσω της μεθόδου `updateClues`, η οποία αναλύει τη νέα κατάσταση του πλέγματος και υπολογίζει ξανά τις ενδείξεις.

```
final List<int> rowClues = rowMatches.map((RegExpMatch match) =>
    match.group(0)!.length).toList();
newVerticalClues[row] = rowClues.isNotEmpty ? rowClues : <int>[0];
```

10.5.5 Ανανέωση Κατάστασης Κουτιού

Η μέθοδος `updateBox` του `CreateNonogramCubit` διαχειρίζεται την ενημέρωση ενός κουτιού. Όταν ένα κουτί πατηθεί ή σύρεται, η κατάστασή του αλλάζει, και οι αλλαγές αντικατοπτρίζονται τόσο στη λύση όσο και στις ενδείξεις.

10.5.5.1 Η Διαδικασία Ενημέρωσης είναι η εξής:

1. **Αναγνώριση Κατάστασης Κουτιού:** Το τρέχον *state* του κουτιού ανιχνεύεται μέσω του χαρακτήρα στην αντίστοιχη θέση της συμβολοσειράς λύσης (`'1'`, `'?'`).
2. **Ενημέρωση Κατάστασης:** Ανάλογα με το *state*, το κουτί γεμίζεται ή διαγράφεται.
3. **Αυτόματη Ενημέρωση Ενδείξεων:** Επανυπολογίζονται οι ενδείξεις της αντίστοιχης γραμμής και στήλης.

10.5.6 Ανανέωση Κατάστασης Στοιχείων

Η αυτόματη ανανέωση των στοιχείων εξασφαλίζει ότι το πλέγμα και οι ενδείξεις παραμένουν συγχρονισμένα. Οι μέθοδοι `updateHorizontalClues` και `updateVerticalClues` χρησιμοποιούν κανονικές εκφράσεις για να ανιχνεύσουν συνεχόμενες ακολουθίες γεμισμένων κουτιών και να ενημερώσουν τις αντίστοιχες ενδείξεις.

Οι ακολουθίες γεμισμένων κουτιών αναγνωρίζονται μέσω του *regex* `RegExp(r'1+')`, που ταιριάζει με μία ή περισσότερες συνεχόμενες `'1'`. Οι θέσεις των ακολουθιών μετατρέπονται σε αριθμούς, οι οποίοι αντιστοιχούν στις νέες ενδείξεις.

10.6 Προσαρμογή Μεγέθους Παζλ

10.6.1 Εισαγωγή

Η προσαρμογή του μεγέθους του παζλ επιτρέπει στους χρήστες να αλλάξουν τις διαστάσεις του πλέγματος, προσθέτοντας ή αφαιρώντας γραμμές και στήλες. Αυτή η λειτουργία υλοποιείται μέσω *sliders* και συνοδεύεται από αυτόματη ανανέωση των κουτιών και των ενδείξεων, ώστε το πλέγμα να παραμένει συνεπές και χρηστικό.

10.6.2 Με τους Sliders

Τα *sliders* ελέγχουν τις διαστάσεις του πλέγματος:

- Οριζόντιο *slider*: Αλλάζει τον αριθμό των στηλών.
- Κάθετο *slider*: Αλλάζει τον αριθμό των γραμμών.

Το κάθε *slider* συνδέεται με τις μεθόδους `updateWidth` και `updateHeight` του `CreateNonogramCubit` για την ανανέωση των διαστάσεων. Οι αλλαγές αντικατοπτρίζονται άμεσα στο *UI*.

```
Slider(
  value: state.width + 0.0,
  min: 1,
  max: _maxSizeValue + 0.0,
  divisions: _maxSizeValue,
  onChanged: (double value) {
    createNonogramCubit.updateWidth(value.ceil());
  },
);
```

10.6.3 Ανανέωση Κουτιών με τις Αλλαγές

10.6.3.1 Εισαγωγή

Η ανανέωση των κουτιών του πλέγματος κατά την αλλαγή του μεγέθους είναι μία βασική λειτουργικότητα που απαιτεί διαφορετική διαχείριση για την αλλαγή του ύψους και του πλάτους. Ενώ η ανανέωση των κουτιών όταν αλλάζει το ύψος είναι σχετικά απλή, η αλλαγή του πλάτους απαιτεί πιο σύνθετη επεξεργασία λόγω της φύσης της διάταξης των στοιχείων στη λύση.

10.6.3.2 Αλλαγή Ύψους

Η προσθήκη νέων γραμμών όταν το ύψος αυξάνεται γίνεται με απλό τρόπο: νέες θέσεις προστίθενται στο τέλος της συμβολοσειράς λύσης.

```
emit(state.copyWith(solution: state.solution + '?' * (totalBoxes - state.solution.length)));
```

Όταν το ύψος μειώνεται, το πλεονάζον μήκος αφαιρείται απλώς από το τέλος:

```
final String newSolution = oldSolution.substring(0, totalBoxes);
emit(state.copyWith(solution: newSolution));
```

Αυτή η διαδικασία διατηρεί τη δομή της λύσης σταθερή, καθώς τα στοιχεία που ανήκουν στις γραμμές παραμένουν στις ίδιες θέσεις.

10.6.3.3 Αλλαγή Πλάτους

10.6.3.3.1 Εισαγωγή

Η ανανέωση των κουτιών κατά την αλλαγή του πλάτους απαιτεί την αναδιοργάνωση της λύσης, επειδή το πλάτος επηρεάζει το σημείο όπου κάθε γραμμή "σπάει" (*wrap*). Όταν το πλάτος αυξάνεται ή μειώνεται, οι θέσεις των στοιχείων πρέπει να αναδιαταχθούν ώστε να ευθυγραμμιστούν με το νέο πλάτος.

10.6.3.3.2 Προσθήκη Στήλης

Όταν το πλάτος αυξάνεται, νέες θέσεις προστίθενται μεταξύ των γραμμών:

```
emit(
  state.copyWith(
    solution: state.solution.replaceAllMapped(
      RegExp(r'.{' + (index - difference).toString() + r'}'),
      (Match match) => "${match.group(0)}${'?' * difference}",
    ),
  ),
);
```

Εδώ, χρησιμοποιείται ένας κανονικός εκφραστής (*RegExp*) για να εντοπιστεί κάθε γραμμή στο *string* και να προστεθούν τα απαραίτητα νέα κουτιά (?) στο τέλος της.

10.6.3.3.3 Αφαίρεση Στήλης

Όταν το πλάτος μειώνεται, τα πλεονάζοντα κουτιά αφαιρούνται από κάθε γραμμή:

```
final String newSolution = oldSolution.replaceAllMapped(
  RegExp(r'(.{' + index.toString() + r'})(.{' + (extraBoxes / state.height).ceil().toString() + r'})
  (Match match) => match.group(1)!,
);
emit(state.copyWith(solution: newSolution));
```

```
K      10
```

Σε αυτή την περίπτωση, ο κανονικός εκφραστής εντοπίζει τις γραμμές και αφαιρεί τα κουτιά που περισσεύουν από το τέλος κάθε γραμμής.

10.6.3.3.4 Διαχείριση Ενδείξεων

Μετά από κάθε αλλαγή στο μέγεθος, ενημερώνονται οι ενδείξεις (*clues*) για να ανταποκρίνονται στη νέα διάταξη:

```
if (newSolution.sumFilledBoxes < oldSolution.sumFilledBoxes) {
  updateHorizontalClues();
  updateVerticalClues();
}
```

Αυτό διασφαλίζει ότι οι ενδείξεις παραμένουν συνεπείς με την ενημερωμένη κατάσταση της λύσης.

10.6.3.4 Συμπεράσματα

Η ανανέωση κουτιών κατά την αλλαγή μεγέθους του πλέγματος εξασφαλίζει τη συνοχή της λύσης, διατηρώντας τη λειτουργικότητα του **Εικονόσταυρου (Nonogram)**. Η διαφορετική προσέγγιση για το ύψος και το πλάτος είναι απαραίτητη για να διαχειριστεί σωστά τη διάταξη των στοιχείων, ενώ οι τεχνικές όπως οι κανονικές εκφράσεις επιτρέπουν την αποδοτική επεξεργασία των δεδομένων.

10.6.4 Ανανέωση Στοιχείων με τις Αλλαγές

10.6.4.1 Εισαγωγή

Η ανανέωση των στοιχείων του παζλ, είτε οριζόντιων είτε κάθετων, αποτελεί κρίσιμη διαδικασία στη λειτουργικότητα του δημιουργού **Εικονόσταυρου (Nonogram)**. Αυτή η λειτουργία εξασφαλίζει ότι τα στοιχεία αντικατοπτρίζουν με ακρίβεια την τρέχουσα κατάσταση του πλέγματος κάθε φορά που πραγματοποιείται αλλαγή στο μέγεθος του πλέγματος ή στην κατανομή των γεμισμένων κουτιών. Η ανάγκη για διαχωρισμό των μεθόδων ανανέωσης οριζόντιων και κάθετων στοιχείων προκύπτει από τη διαφορετική φύση των δεδομένων τους: τα οριζόντια στοιχεία αντιστοιχούν σε συνεχόμενα τμήματα της συμβολοσειράς του `solution`, ενώ τα κάθετα εξάγονται από διάσπαρτα τμήματα.

Η ενότητα αυτή παρουσιάζει τη διαδικασία και τις μεθόδους που χρησιμοποιούνται για την ανανέωση των στοιχείων, δίνοντας έμφαση στις τεχνικές προκλήσεις που αντιμετωπίζονται κατά τη διαχείριση των κάθετων δεδομένων σε σχέση με τα οριζόντια.

10.6.4.2 Ανανέωση Οριζόντιων Στοιχείων

Η ανανέωση των οριζόντιων στοιχείων (`updateHorizontalClues`) βασίζεται στη δομή του `solution string`, καθώς κάθε γραμμή αντιστοιχεί σε συνεχόμενο τμήμα της συμβολοσειράς. Ο αλγόριθμος επεξεργάζεται κάθε γραμμή ξεχωριστά, εξάγοντας τα δεδομένα της μέσω της μεθόδου `getRowIsolate`. Η μέθοδος αυτή επιστρέφει μια συμβολοσειρά που περιλαμβάνει όλα τα κουτιά της γραμμής, τα οποία είναι συνεχόμενα στο `solution string` λόγω της οριζόντιας διάταξης.

Για κάθε γραμμή, εφαρμόζεται ο κανονικός εκφραστής `RegExp(r'1+')` για την αναγνώριση συνεχόμενων συνόλων γεμισμένων κουτιών (1). Τα μήκη αυτών των συνόλων υπολογίζονται και αποθηκεύονται στη λίστα των οριζόντιων στοιχείων.

```
final Iterable<RegExpMatch> rowMatches = regExp.allMatches(
  state.solution.getRowIsolate(row, state.width)
);
final List<int> rowClues = rowMatches.map((RegExpMatch match) => match.group(0)!.length).toList();
newVerticalClues[row] = rowClues.isNotEmpty ? rowClues : <int>[0];
```

Στο τέλος της διαδικασίας, η κατάσταση του πλέγματος ενημερώνεται μέσω της μεθόδου `emit`. Αυτό διασφαλίζει ότι οι αλλαγές αποθηκεύονται και απεικονίζονται άμεσα στη διεπαφή χρήστη.

10.6.4.3 Ανανέωση Κάθετων Στοιχείων

Η ανανέωση των κάθετων στοιχείων (`updateVerticalClues`) είναι πιο σύνθετη, καθώς τα δεδομένα για κάθε στήλη πρέπει να ανακτηθούν από διάσπαρτα τμήματα της συμβολοσειράς. Η μέθοδος `getColumnIsolate` χρησιμοποιείται για την εξαγωγή δεδομένων της στήλης, επιστρέφοντας μια νέα συμβολοσειρά που περιλαμβάνει όλα τα κουτιά της στήλης.

Ο αλγόριθμος εφαρμόζει τον ίδιο κανονικό εκφραστή (`RegExp(r'1+')`) στις εξαγόμενες στήλες για την αναγνώριση συνεχόμενων συνόλων γεμισμένων κουτιών. Τα μήκη αυτών των συνόλων υπολογίζονται και αποθηκεύονται στη λίστα των κάθετων στοιχείων.

```
final Iterable<RegExpMatch> columnMatches = regExp.allMatches(
  state.solution.getColumnIsolate(column, state.width)
);
final List<int> columnClues = columnMatches.map((RegExpMatch match) =>
    match.group(0)!.length).toList();
newHorizontalClues[column] = columnClues.isNotEmpty ? columnClues : <int>[0];
emit(state.copyWith(horizontalClues: newHorizontalClues));
```

10.6.4.4 Διαφορά Διαχείρισης Οριζόντιων και Κάθετων Στοιχείων

Ενώ τα οριζόντια στοιχεία μπορούν να υπολογιστούν απευθείας από συνεχόμενα τμήματα του `solution string`, τα κάθετα απαιτούν πιο περίπλοκη λογική, καθώς η εξαγωγή τους απαιτεί τη συλλογή δεδομένων από μη συνεχόμενα μέρη. Αυτή η διαφοροποίηση καθιστά τη διαχείριση των κάθετων στοιχείων πιο χρονοβόρα και υπογραμμίζει την ανάγκη για εξειδικευμένες μεθόδους για την αποτελεσματική τους επεξεργασία.

10.7 Συμπεράσματα

Η ενότητα για τη δημιουργία και διαχείριση ενός Εικονόσταυρου παρέχει μια ολοκληρωμένη εικόνα της πολυπλοκότητας και της ευελιξίας του συστήματος που αναπτύχθηκε. Το σύστημα επιτρέπει στους **χρήστες** να σχεδιάζουν και να επεξεργάζονται παζλ μέσω διαισθητικών *εργαλείων* και δυναμικής προσαρμογής του πλέγματος, προσφέροντας μια φιλική προς τον χρήστη εμπειρία, χωρίς να μειώνεται η λειτουργικότητα ή η ακρίβεια του λογισμικού.

Η *δυνατότητα αλλαγής* του μεγέθους του πλέγματος, τόσο στο ύψος όσο και στο πλάτος, αποτελεί ένα από τα βασικότερα χαρακτηριστικά της εφαρμογής. Ενώ η ανανέωση των στοιχείων για την αλλαγή ύψους είναι σχετικά απλή, καθώς προστίθενται ή αφαιρούνται κουτιά στο τέλος της συμβολοσειράς, η ανανέωση των δεδομένων για την αλλαγή πλάτους απαιτεί πιο πολύπλοκη επεξεργασία. Οι αλλαγές στο πλάτος προκαλούν μετακίνηση των κουτιών μέσα στη συμβολοσειρά λόγω της αλλαγής που δημιουργείται στην *αλλαγή γραμμής*, γεγονός που επιλύεται μέσω προηγμένων τεχνικών χειρισμού συμβολοσειρών.

Η ανανέωση των στοιχείων του πλέγματος παρουσιάζει ιδιαίτερο ενδιαφέρον, καθώς τα δεδομένα οριζόντιων και κάθετων στοιχείων αντιμετωπίζονται διαφορετικά. Τα οριζόντια στοιχεία, αποτελώντας συνεχόμενα τμήματα της *συμβολοσειράς λύσης*, εξάγονται πιο εύκολα μέσω κανονικών εκφράσεων. Αντίθετα, τα κάθετα στοιχεία απαιτούν την επεξεργασία δεδομένων από διάσπαρτα σημεία της συμβολοσειράς, γεγονός που επιβάλλει πιο σύνθετη λογική. Οι μέθοδοι `updateHorizontalClues` και `updateVerticalClues` καταδεικνύουν αυτήν τη διαφορά και υπογραμμίζουν τη σημασία της προσεκτικής διαχείρισης των δεδομένων σε διαφορετικούς άξονες.

Η προσθήκη λειτουργιών όπως η ενημέρωση των κουτιών μέσω *χειρονομιών*, η δυναμική προσαρμογή των στοιχείων κατά τη σχεδίαση και η εύκολη εισαγωγή ενδείξεων μέσω πεδίων κειμένου ενισχύουν την ευχρηστία του εργαλείου. Παράλληλα, η υλοποίηση αυτών των λειτουργιών με αποδοτικούς αλγόριθμους και έξυπνη διαχείριση της κατάστασης διασφαλίζει τη σταθερότητα και την απόδοση της εφαρμογής, ακόμη και σε μεγάλα πλέγματα.

Συνολικά, η σελίδα δημιουργίας Εικονόσταυρου επιτυγχάνει έναν συνδυασμό απλότητας και πολυπλοκότητας: προσφέρει στους **χρήστες** τα μέσα να δημιουργούν παζλ με ελευθερία και ακρίβεια, ενώ παράλληλα αποκρύπτει την υπολογιστική πολυπλοκότητα που απαιτείται για την ομαλή λειτουργία του συστήματος. Οι αρχές σχεδιασμού και οι αλγοριθμικές λύσεις που παρουσιάστηκαν σε αυτήν την ενότητα λειτουργούν ως οδηγός για τη δημιουργία δυναμικών και ευέλικτων συστημάτων, τα οποία διασφαλίζουν ποιοτική εμπειρία στους χρήστες.

Κεφάλαιο 11ο: Προκλήσεις και Λύσεις

11.1 Εισαγωγή

Κατά τη διάρκεια της ανάπτυξης του λύτη Εικονόσταυρων, η πορεία ήταν γεμάτη προκλήσεις που δοκίμασαν τόσο την τεχνική υλοποίηση όσο και την ιδεολογική βάση της εφαρμογής. Αυτές οι προκλήσεις δεν αποτέλεσαν απλώς εμπόδια, αλλά και ευκαιρίες για βαθύτερη κατανόηση του αντικειμένου και για την ανάπτυξη λύσεων που βελτίωσαν την απόδοση, τη λειτουργικότητα και τη συνολική εμπειρία του χρήστη.

Η ανάπτυξη του λύτη ξεκίνησε με έναν απλό στόχο: να δημιουργηθεί μια εφαρμογή που θα επιλύει αποτελεσματικά τα Εικονόσταυρα. Ωστόσο, η πραγμάτωση αυτού του στόχου αποδείχθηκε πιο περίπλοκη από ό,τι αρχικά υπολογιζόταν. Από την ιδέα μέχρι την πράξη, παρουσιάστηκαν πολλά εμπόδια που απαιτούσαν δημιουργική σκέψη και ενδελεχή ανάλυση για να ξεπεραστούν. Η αρχική ιδεολογία του αλγορίθμου επίλυσης, οι τεχνικοί περιορισμοί της **Flutter**, η ανάγκη για ταχύτητα και απόδοση, η συμβατότητα με διαφορετικά περιβάλλοντα, και η βελτίωση της εμπειρίας του χρήστη ήταν μόνο μερικά από τα πεδία όπου εμφανίστηκαν σημαντικές προκλήσεις.

Η ενότητα αυτή εξετάζει τις σημαντικότερες δυσκολίες που συναντήθηκαν κατά την εξέλιξη του έργου, από τα πρώτα βήματα σχεδιασμού του αλγορίθμου μέχρι την ολοκλήρωση της εφαρμογής. Περιγράφονται οι λύσεις που εφαρμόστηκαν, οι οποίες συνδύασαν τεχνικές βελτιστοποίησης, επανασχεδιασμό του αλγορίθμου, και υιοθέτηση νέων τεχνολογιών και προσεγγίσεων. Παράλληλα, αναλύεται πώς οι λύσεις αυτές όχι μόνο επέλυσαν τα εκάστοτε εμπόδια, αλλά συνέβαλαν και στην αναβάθμιση της λειτουργικότητας και της καινοτομίας του λογισμικού.

Στα επόμενα τμήματα, καταγράφονται συγκεκριμένες προκλήσεις και λύσεις που αφορούν τη θεμελιώδη λογική του αλγορίθμου, τη διαχείριση της πολυπλοκότητάς του, τη βελτιστοποίηση της απόδοσης, την αντιμετώπιση ζητημάτων συμβατότητας με φυλλομετρητές και διάφορες πλατφόρμες, την ενσωμάτωση πολυνηματικών προσεγγίσεων μέσω *workers*, και τη διαμόρφωση ενός ολοκληρωμένου περιβάλλοντος για τη δημιουργία και επεξεργασία παζλ. Αυτή η ανασκόπηση δεν εστιάζει μόνο στις τεχνικές λεπτομέρειες, αλλά αναδεικνύει επίσης τη στρατηγική σκέψη και την ευελιξία που χρειάστηκαν για την επιτυχή ολοκλήρωση του έργου.

11.2 Δημιουργία Αρχικής Ιδεολογίας

Στην αρχική φάση σχεδιασμού του λύτη, η ιδεολογία στράφηκε στο να μιμηθεί τον τρόπο με τον οποίο ένας άνθρωπος προσεγγίζει και λύνει ένα Εικονόσταυρο. Η προσέγγιση αυτή βασιζόταν σε τεχνικές που χρησιμοποιούν οι άνθρωποι, όπως η αξιολόγηση των πιο υποσχόμενων γραμμών ή στηλών, και η αξιοποίηση εμπειρικών κανόνων για τη συμπλήρωση των κουτιών. Ωστόσο, η στρατηγική αυτή αποδείχθηκε περιοριστική για έναν αλγόριθμο, διότι οι άνθρωποι διαθέτουν μια φυσική ικανότητα επεξεργασίας μεγάλου όγκου πληροφοριών ταυτόχρονα, ενώ ο υπολογισμός απαιτεί αυστηρές, σαφείς και δομημένες διαδικασίες ώστε να είναι αποδοτικός.

Η μετάβαση από μια “ανθρώπινη” θεώρηση σε μια αυστηρά “υπολογιστική” προσέγγιση κατέστη απαραίτητη για την αντιμετώπιση των περιορισμών που ανέκυπταν. Παρότι οι άνθρωποι μπορούν να κάνουν υποκειμενικές κρίσεις με βάση τη συνολική όψη ενός πλέγματος, ένας αλγόριθμος χρειάζεται συγκεκριμένους κανόνες και βήματα για να διασφαλίζει ακρίβεια και αποτελεσματικότητα. Για παράδειγμα, ένας άνθρωπος δεν θα ελέγξει ποτέ κάθε κουτάκι ξεχωριστά για κάθε ένδειξη, ούτε θα διατηρήσει μια ολοκληρωμένη λίστα με τις επόμενες γραμμές ή στήλες προς ανάλυση. Η ικανότητα ενός ανθρώπου να “ρίχνει μια ματιά” σε ένα πλέγμα και να αποφασίζει ποια γραμμή είναι υποσχόμενη, χωρίς να βασίζεται αυστηρά σε μαθηματική ή προγραμματιστική λογική, δεν μεταφέρεται εύκολα σε έναν αλγόριθμο.

Ως εκ τούτου, η λογική της επίλυσης προσανατολίστηκε στην πλήρη αξιοποίηση των δεδομένων μέσω συστηματικής ανάλυσης. Ο αλγόριθμος σχεδιάστηκε ώστε να εξετάζει μεθοδικά κάθε κουτάκι και κάθε ένδειξη, δημιουργώ-

ντας μια δομή που εξασφαλίζει την αξιοποίηση του συνόλου των διαθέσιμων πληροφοριών. Αντί να "προβλέπει" σε ποια γραμμή ή στήλη πρέπει να εστιάσει, ο αλγόριθμος στηρίζεται σε μαθηματικούς κανόνες ώστε να επιλέγει τα επόμενα βήματα.

Η αλλαγή αυτή στην ιδεολογία επέτρεψε στον λύτη να αξιοποιήσει πλήρως τις δυνατότητες του υπολογιστή και να ξεπεράσει τις αδυναμίες της καθαρά ανθρώπινης προσέγγισης. Παρόλο που ένα λογισμικό δεν μπορεί να "δει" ένα πλέγμα με τον ίδιο τρόπο που το αντιλαμβάνεται ένας άνθρωπος, μπορεί να εκτελέσει μια ακολουθία λογικών βημάτων που διασφαλίζουν ότι συλλέγονται και αξιοποιούνται όλες οι διαθέσιμες πληροφορίες. Αυτή η στροφή δεν αποτέλεσε μόνο ανάγκη, αλλά ήταν και καθοριστική για την επιτυχία της εφαρμογής, καθώς ενίσχυσε την ακρίβεια και την επαναληψιμότητα του αποτελέσματος.

11.3 Έλεγχος Αποτελεσμάτων του Λύτη

Καθ' όλη τη φάση ανάπτυξης, μία από τις μεγαλύτερες προκλήσεις ήταν η κατανόηση και ο έλεγχος της λειτουργίας του λύτη, ιδιαίτερα καθώς ο αλγόριθμος γινόταν ολοένα πιο περίπλοκος. Με τη σταδιακή προσθήκη νέων λειτουργιών και βελτιώσεων, η απλή παρατήρηση του τελικού αποτελέσματος δεν επαρκούσε για να εξασφαλίσει ότι ο λύτης λειτουργούσε ορθά. Για τον λόγο αυτό, ενισχύθηκε η διαδικασία **καταγραφής**, η οποία παρείχε λεπτομερή εκτύπωση της πορείας επίλυσης, μαζί με κρίσιμες μεταβλητές και ενδιάμεσα αποτελέσματα.

Αυτή η συνεπής καταγραφή επέτρεψε τη σαφή χαρτογράφηση της ροής του αλγορίθμου. Σε κάθε κρίσιμο σημείο λήψης απόφασης, εκτυπωνόταν το αποτέλεσμα της ανάλυσης, μαζί με τα δεδομένα που χρησιμοποιήθηκαν. Οι ενδιάμεσες λύσεις και οι μεταβολές σε αυτές καταγράφονταν συστηματικά, ώστε να διαπιστώνεται αν το λογισμικό λειτουργούσε βάσει της αναμενόμενης λογικής. Για παράδειγμα, όταν ελεγχόταν η πιθανή τοποθέτηση μιας ένδειξης σε ένα συγκεκριμένο σημείο, εκτυπωνόταν η θέση που εξετάστηκε, η εξεταζόμενη ένδειξη, καθώς και το αν η τοποθέτηση θεωρήθηκε εφικτή.

Επιπλέον, μετά από κάθε αλλαγή, αποθηκευόταν η τρέχουσα κατάσταση του πλέγματος της λύσης, προσφέροντας μια δυναμική οπτικοποίηση της εξέλιξης του αλγορίθμου. Όταν συμπληρωνόταν ένα κουτάκι, καταγραφόταν το ποιο ήταν, ποια ένδειξη το επηρέασε, και πώς επηρέασε το συνολικό πλέγμα. Με αυτό τον τρόπο, γινόταν εφικτή η συνεχής εποπτεία της διαδικασίας επίλυσης, καθιστώντας δυνατή την άμεση ανίχνευση και αντιμετώπιση λαθών ή απρόβλεπτων καταστάσεων.

Η μεθοδολογία αυτή συνέβαλε στον έλεγχο της συμφωνίας των δεδομένων με τη θεωρητική λογική του αλγορίθμου. Η συνεχής εκτύπωση των αποτελεσμάτων και των συμπερασμάτων κάθε βήματος προσέφερε μια πλήρη εικόνα της εκτέλεσης, αποτελώντας ουσιαστικό εργαλείο για τη διασφάλιση ποιότητας, τη βελτιστοποίηση της εφαρμογής και την επιβεβαίωση ότι ο λύτης λειτουργούσε σύμφωνα με τις απαιτήσεις.

11.4 Συμβατότητα με Safari και iOS

11.4.1 Εισαγωγή

Η ανάπτυξη λογισμικού που να λειτουργεί ομαλά σε όλα τα περιβάλλοντα και *browsers* αποτελεί μια σημαντική πρόκληση, ιδιαίτερα όταν χρησιμοποιούνται τεχνολογίες με διαφορετικά επίπεδα υποστήριξης. Ένα χαρακτηριστικό ζήτημα που εμφανίστηκε κατά την εξέλιξη της εφαρμογής ήταν η ασυμβατότητα του αλγορίθμου επίλυσης με τον *Safari* και τους *browsers* σε *iOS* συσκευές. Παρόλο που η εφαρμογή λειτούργησε σωστά σε *Chrome*, *Firefox*, *Edge* και σε συσκευές *Android*, τα αποτελέσματα στον *Safari* και σε όλους τους *browsers* που χρησιμοποιούν υποχρεωτικά *WebKit* σε *iOS* ήταν λανθασμένα.

Το πρόβλημα εντοπίστηκε στην αξιοποίηση του *lookbehind* στις *regular expressions* του αλγορίθμου. Τα *lookbehinds*

είναι ιδιαίτερα χρήσιμα για την αναζήτηση ενός *pattern* πριν από ένα σημείο ενδιαφέροντος, όμως δεν υποστηρίζονται από όλους τους *browsers*. Ειδικότερα, απαιτούν τον κινητήρα *JavaScript V8* που χρησιμοποιείται, μεταξύ άλλων, από *Chrome* και *Edge*. Αντίθετα, ο κινητήρας *WebKit*, που υποχρεωτικά χρησιμοποιείται σε όλες τις *iOS* συσκευές, δεν υποστηρίζει αυτήν τη λειτουργία, με αποτέλεσμα οι αντίστοιχες κανονικές εκφράσεις να αποτυγχάνουν.

11.4.2 Τεχνική Διάγνωση του Προβλήματος

Η αρχική υλοποίηση του αλγορίθμου βασιζόταν σε *regex* με *lookbehind* για την ενημέρωση της λύσης ενός παζλ. Ο αλγόριθμος χρησιμοποιούσε *lookbehind* ώστε να εντοπίζει τη θέση συγκεκριμένων χαρακτήρων σε γραμμές ή στήλες του παζλ και να τους αντικαθιστά με τις κατάλληλες τιμές (0 ή 1). Αυτή η προσέγγιση ήταν απλή και αποτελεσματική, αλλά δεδομένου ότι το *lookbehind* δεν υποστηρίζεται από το *WebKit*, ο *Safari* και οι *browsers* σε *iOS* αδυνατούσαν να εκτελέσουν τον κώδικα ορθά, παράγοντας λανθασμένες λύσεις μόνο σε αυτά τα περιβάλλοντα.

Η διερεύνηση του προβλήματος έδειξε ότι το *lookbehind* εφαρμοζόταν για να εντοπιστούν χαρακτήρες βάσει ενός ορισμένου *pattern* με συγκεκριμένο αριθμό χαρακτήρων πριν από τη θέση τους. Για παράδειγμα, ο αλγόριθμος μπορούσε να αντικαταστήσει όλα τα "?" στις αντίστοιχες θέσεις ενός *string* λύσης με "0", εφόσον αυτές οι θέσεις προσδιορίζονταν από τη σχετική *regex*. Όμως, επειδή η υλοποίηση εξαρτιόταν απόλυτα από το *lookbehind*, απέκλειε τη σωστή λειτουργία σε *Safari*.

11.4.3 Λύση και Ανασχεδιασμός

Για να διασφαλιστεί η συμβατότητα, ο αλγόριθμος ανασχεδιάστηκε ώστε να αποφεύγει πλήρως το *lookbehind*. Η νέα λύση βασίστηκε στην αξιοποίηση **substrings**. Συγκεκριμένα, το *string* της λύσης χωρίζεται σε δύο τμήματα: το πρώτο περιλαμβάνει όλους τους χαρακτήρες μέχρι το σημείο που πρέπει να αλλαχθεί, ενώ το δεύτερο περιέχει τους χαρακτήρες που ακολουθούν. Κατόπιν, ο νέος χαρακτήρας (0 ή 1) τοποθετείται ανάμεσα σε αυτά τα δύο *substrings*, δημιουργώντας ένα ενημερωμένο *string* λύσης. Έτσι, επιτυγχάνεται η ίδια λειτουργία αντικατάστασης χωρίς την ανάγκη *lookbehind*.

Η αλλαγή αυτή ολοκληρώθηκε με τη δημιουργία μιας ξεχωριστής μεθόδου, της `getFilledInSolution`, η οποία επιτρέπει την ταχεία και ευέλικτη ενημέρωση της λύσης βάσει των υπολογισμών του αλγορίθμου. Περιγράφεται αναλυτικά στην ενότητα Κεφάλαιο 9.5.5.1.3.1 *Περιγραφή Λειτουργίας της getFilledInSolution* και αποτελεί τον πυρήνα της νέας προσέγγισης για την αντικατάσταση χαρακτήρων.

11.4.4 Εργασία της Μεθόδου

Ο πρωταρχικός ρόλος τόσο της αρχικής υλοποίησης με *lookbehind* όσο και της νέας προσέγγισης είναι η ενημέρωση της λύσης βάσει των νέων ενδείξεων. Οι πληροφορίες των ενδείξεων απαιτείται να αντιστοιχηθούν στις κατάλληλες θέσεις του *string* της λύσης και να αντικατασταθούν με τις αντίστοιχες τιμές. Η αρχική στρατηγική αξιοποιούσε τις δυνατότητες απλής αναζήτησης και αντικατάστασης χαρακτήρων που προσφέρουν οι *regex* με *lookbehind*. Ωστόσο, η μέθοδος που βασίζεται στα *substrings* υλοποιεί την ίδια λογική με τη βοήθεια στοιχειωδών πράξεων και είναι απόλυτα συμβατή με το *WebKit*.

11.4.5 Συμπεράσματα

Η μετάβαση από *lookbehind* σε *substrings* διασφάλισε την απρόσκοπτη λειτουργία της εφαρμογής στον *Safari* και σε όλες τις *iOS* συσκευές, διατηρώντας ταυτόχρονα την απόδοση και την ευελιξία του αλγορίθμου. Το συγκεκριμένο ζήτημα ανέδειξε τη σημασία κατανόησης των τεχνολογικών περιορισμών και της εφαρμογής λύσεων

που είναι διαλειτουργικές, ενώ η εμπειρία που αποκτήθηκε συνέβαλε στην ενίσχυση της ανθεκτικότητας και της επεκτασιμότητας του λογισμικού.

11.5 Βελτιστοποίηση Αλγορίθμου

11.5.1 Εισαγωγή

Η βελτιστοποίηση του αλγορίθμου επίλυσης αποτέλεσε μία από τις πλέον απαιτητικές και καθοριστικές διαδικασίες κατά την ανάπτυξη της εφαρμογής. Από τις αρχικές εκδόσεις έως την τελική μορφή, ο αλγόριθμος παρουσίασε σημαντική εξέλιξη τόσο ως προς την *ταχύτητα* όσο και ως προς την **αποδοτικότητα** στην κατανάλωση πόρων. Αυτές οι βελτιώσεις κατέστησαν τον αλγόριθμο ικανό να διαχειρίζεται μεγαλύτερα και πιο σύνθετα παζλ, ενώ παράλληλα προσέφεραν μια πιο άμεση και ευχάριστη εμπειρία στον χρήστη.

11.5.2 Αρχικές Προκλήσεις

Στις πρώτες φάσεις της ανάπτυξης, ο αλγόριθμος αντιμετώπιζε σημαντικές δυσκολίες όσον αφορά την απόδοση. Για τα ιδιαίτερα απαιτητικά παζλ, ο χρόνος επίλυσης μπορούσε να φτάσει ή και να υπερβεί τα οκτώ λεπτά, γεγονός μη αποδεκτό από άποψη εμπειρίας χρήστη αλλά και συνολικής αξιολόγησης του αλγόριθμου. Παράλληλα, κατά τη διάρκεια της επίλυσης, η εφαρμογή **πάγωνε**, επειδή ο ενσωματωμένος μηχανισμός ενημέρωσης της οθόνης δεν εκτελούνταν έγκαιρα. Η *Flutter*, ως **μονονηματικό** περιβάλλον (*single-threaded framework*), δεν υποστηρίζει ταυτόχρονες εργασίες, με αποτέλεσμα η διεπαφή να παραμένει αδρανής μέχρι να ολοκληρωθεί η εκτέλεση του αλγορίθμου.

11.5.3 Προβλήματα Υλοποίησης

Τα αίτια της βραδύτητας του αλγορίθμου εντοπίστηκαν κυρίως στη χρήση πολλαπλών *for loops* και στην αναδρομική του φύση. Παρόλο που οι λύσεις αυτές ήταν σχετικά απλές ως σύλληψη, οδηγούσαν σε **εκθετική** αύξηση της πολυπλοκότητας με την αύξηση του μεγέθους του παζλ. Επιπλέον, σε κάθε επανάληψη των βρόχων επανεξετάζονταν δεδομένα που είχαν ήδη υπολογιστεί, προκαλώντας περιττό φόρτο επεξεργασίας.

11.5.4 Εντοπισμός Σημείων Βελτίωσης

Η ενδελεχής ανάλυση του κώδικα έφερε στο φως συγκεκριμένα προβληματικά σημεία που θα μπορούσαν να βελτιστοποιηθούν. Ένα βασικό παράδειγμα ήταν η *αναγκαστική επανεξέταση* δεδομένων που είχαν ήδη χρησιμοποιηθεί. Κάθε στήλη και γραμμή του παζλ ελεγχόταν επανειλημμένα, ακόμη και όταν δεν υπήρχε λόγος. Επιπρόσθετα, η απουσία **caching** ανάγκαζε τον αλγόριθμο να εκτελεί ξανά και ξανά τους ίδιους υπολογισμούς.

11.5.5 Υλοποίηση Βελτιστοποιήσεων

Η πρώτη σημαντική βελτίωση επικεντρώθηκε στην προσθήκη μηχανισμών **caching**. Τα δεδομένα που προέκυπταν από συγκεκριμένους υπολογισμούς αποθηκεύονταν προσωρινά, έτσι ώστε να επανεξετάζονται μόνο όταν κρινόταν απαραίτητο. Με αυτόν τον τρόπο, μειώθηκαν δραστικά οι *περιττοί υπολογισμοί* και επιτεύχθηκε καλύτερη διαχείριση της μνήμης.

Ακολούθως, εγκαταλείφθηκε η *αναδρομική* προσέγγιση και υιοθετήθηκαν επαναληπτικές μέθοδοι. Οι βρόχοι αναστοχεδιάστηκαν, με σκοπό να αποφευχθεί η εκθετική αύξηση της πολυπλοκότητας. Βήμα προς βήμα, απομακρύνθηκαν τα αναδρομικά στοιχεία, επιτυγχάνοντας πιο ευέλικτη και αποδοτική ροή επεξεργασίας.

11.5.6 Μεταφορά σε Πολυνηματική Υλοποίηση (Multithreading)

Για ακόμα μεγαλύτερη βελτίωση της απόδοσης, επιλέχθηκε η πολυνηματική υλοποίηση (*multithreading*). Με τη χρήση του `isolate_manager`, διαμοιράστηκαν τα επιμέρους κομμάτια του αλγορίθμου σε διαφορετικά *threads*. Αυτό επέτρεψε στην **διεπαφή χρήστη** να παραμένει λειτουργική, ενώ η επίλυση εκτελείτο στο παρασκήνιο, αξιοποιώντας πλήρως τους διαθέσιμους πυρήνες του επεξεργαστή. Για να καταστεί αυτό εφικτό, χρειάστηκε αναδιάρθρωση του κώδικα, συμπεριλαμβανομένης της αφαίρεσης οποιουδήποτε *Flutter package* από τους αλγορίθμους, ώστε να μπορούν να λειτουργούν ανεξάρτητα σε *isolates*. Περισσότερες τεχνικές λεπτομέρειες περιγράφονται στο *Κεφάλαιο 11.6 Χρήση Isolates και Workers*.

11.5.7 Μείωση του Χρόνου Επίλυσης

Με τις αναφερόμενες βελτιστοποιήσεις, ο χρόνος για την επίλυση των πλέον απαιτητικών παζλ *μειώθηκε* από περίπου οκτώ λεπτά σε ελάχιστα δευτερόλεπτα. Η χρήση **caching** και **multithreading** όχι μόνο επιτάχυνε την εκτέλεση, αλλά βελτίωσε συνολικά την εμπειρία του χρήστη. Επιπροσθέτως, προστέθηκε μηχανισμός ενημέρωσης για την πορεία της επίλυσης, μέσω μιας *progress bar* που ανανεώνεται σε πραγματικό χρόνο.

11.5.8 Τεχνικές Προσεγγίσεις

Στον βελτιστοποιημένο αλγόριθμο εφαρμόστηκαν και άλλες ειδικές τεχνικές για τη μείωση της πολυπλοκότητας:

- *Αποφυγή Επαναλήψεων*: Τα ήδη συμπληρωμένα κουτάκια δεν ελέγχονται ξανά, περιορίζοντας τις περιττές διεργασίες.
- *Στοχευμένος Έλεγχος*: Γραμμές ή στήλες χωρίς περαιτέρω δυνατή μεταβολή παραλείπονται από την επεξεργασία.
- *Εξαγωγή Μεθόδων*: Η λογική σημαντικών υπολογισμών, όπως η ενημέρωση στοιχείων της λύσης, υλοποιήθηκε σε αυτόνομες μεθόδους, προσφέροντας πιο καθαρή και επεκτάσιμη δομή κώδικα.

11.5.9 Συμπεράσματα

Η εμπειρία βελτιστοποίησης του αλγορίθμου ανέδειξε ότι ακόμα και ένα αρχικά βαρύ και αναποτελεσματικό σύστημα μπορεί, με στοχευμένες διορθώσεις και συνεχή αξιολόγηση, να μετατραπεί σε μια ταχύτατη και λειτουργική λύση. Μέσα από διαδοχικές βελτιώσεις, ο αλγόριθμος ανταποκρίθηκε στις αυξανόμενες απαιτήσεις της εφαρμογής, προσφέροντας μια υψηλής ποιότητας εμπειρία χρήστη και ιδιαίτερα ταχείς χρόνους απόκρισης.

11.6 Χρήση Isolates και Workers

11.6.1 Εισαγωγή

Η ενσωμάτωση *isolates* και *workers* αποτέλεσε καθοριστικό βήμα στην ανάπτυξη της εφαρμογής, καθώς προσέφερε λύση σε ένα από τα μεγαλύτερα ζητήματα: την αναστολή του **UI** κατά την εκτέλεση του αλγορίθμου επίλυσης. Στο *Flutter*, το οποίο λειτουργεί ως **single-threaded framework**, οι βαριές διεργασίες εκτελούνται στο κύριο νήμα (*main thread*), το ίδιο που είναι υπεύθυνο και για την ενημέρωση της διεπαφής χρήστη. Έτσι, όταν η εφαρμογή επεξεργαζόταν μεγάλα και σύνθετα παζλ, εμφάνιζε *πάγωμα*, επιδεινώνοντας την εμπειρία χρήστη.

11.6.2 Αρχική Προσέγγιση και Ανακάλυψη Περιορισμών

Η αρχική προσπάθεια διατήρησης της λειτουργικότητας του **UI** κατά την περίοδο επεξεργασίας βασίστηκε στην εκτέλεση του αλγορίθμου επίλυσης στο κύριο νήμα, συνδυασμένη με μηχανισμούς παρακολούθησης προόδου. Ωστόσο, αυτή η στρατηγική δεν απέδωσε τα αναμενόμενα αποτελέσματα, καθώς το *Flutter* δεν υποστηρίζει πραγματική παράλληλη εκτέλεση στο ίδιο νήμα.

Μέσα από επιπρόσθετη έρευνα, κατέστη σαφές ότι η μόνη ρεαλιστική λύση για τη βελτίωση της απόκρισης του **UI** ήταν η χρήση *isolates*, μια τεχνική της *Dart* που επιτρέπει την εκτέλεση εργασιών σε διαφορετικά νήματα. Παρά ταύτα, αποκαλύφθηκε ότι το *Flutter* δεν υποστηρίζει πλήρως τη χρήση *isolates* σε όλες τις πλατφόρμες, ειδικά στο *Web*.

11.6.3 Αναζήτηση Εναλλακτικών με Workers

Οι περιορισμοί του *Flutter Web* οδήγησαν στην αναζήτηση εναλλακτικών προσεγγίσεων. Σε αυτή τη φάση, εντοπίστηκε το πακέτο `isolate_manager`, το οποίο αξιοποιεί *workers*, μια ελαφρύτερη εκδοχή των *isolates* συμβατή με το *Flutter Web*. Η μετάβαση στους *workers* απαίτησε σημαντικές τροποποιήσεις στον κώδικα, όμως επέτρεψε την εκτέλεση σύνθετων υπολογισμών *εκτός* του κύριου νήματος, εξασφαλίζοντας τη συνεχή ανταπόκριση του **UI**.

11.6.4 Τεχνικές Προκλήσεις και Λύσεις

11.6.4.1 Εισαγωγή

Η υιοθέτηση των *workers* συνεπαγόταν μια σειρά από τεχνικές δυσκολίες, απαιτώντας προσεκτικές αλλαγές στον κώδικα.

11.6.4.2 Αφαίρεση Πακέτων Flutter από τον Αλγόριθμο

Οι *workers* λειτουργούν **εκτός** του πλαισίου του *Flutter*, επομένως δεν μπορούν να χρησιμοποιηθούν *Flutter packages*. Ήταν απαραίτητο να απομακρυνθούν όλα τα *Flutter packages* από τον αλγοριθμικό κώδικα και να αντικατασταθούν με καθαρή *Dart*. Για παράδειγμα, η μέθοδος `characters` που χρησιμοποιείται για την επεξεργασία *Strings* αντικαταστάθηκε από μεθόδους όπως `split('')`.

11.6.4.3 Σειριοποίηση και Αποσειριοποίηση Δεδομένων

Για την επικοινωνία του κύριου νήματος με τους *workers* απαιτείται η αποστολή και λήψη δεδομένων σε **σειριοποιημένη** μορφή (*serialization*). Στο σκέλος της παραλαβής, τα δεδομένα **αποσειριοποιούνται** (*deserialization*). Αυτό το βήμα επέβαλε τη δημιουργία ειδικών δομών δεδομένων για την ομαλή διαχείριση των εισόδων και εξόδων του αλγορίθμου επίλυσης.

11.6.4.4 Αυτονομία του Αλγορίθμου

Ο αρχικός αλγόριθμος βασιζόταν σε κλάσεις *state management* για την ενημέρωση της κατάστασης. Με τη χρήση *workers*, αυτή η προσέγγιση δεν ήταν εφικτή. Ο αλγόριθμος έπρεπε να είναι πλήρως **αυτόνομος**, χειριζόμενος εσωτερικά τις μεταβλητές του και επιστρέφοντας τα αποτελέσματά του ή την πρόοδο που έχει σημειωθεί μέσω μηνυμάτων.

11.6.5 Αναδιάρθρωση και Ενσωμάτωση

Υλοποιώντας όλες τις ανωτέρω αλλαγές, ο αλγόριθμος επανασχεδιάστηκε ώστε να τρέχει πλήρως σε *workers*. Αυτό προσέφερε πολλαπλά πλεονεκτήματα:

- **Αποδοτική Επικοινωνία**: Χάρη στην αξιοποίηση σειριοποιημένων δεδομένων, τα αποτελέσματα μεταφέρονται στο κύριο νήμα με ταχύτητα και αξιοπιστία.
- **Διαχείριση Πόρων**: Η εκτέλεση των απαιτητικών εργασιών σε ξεχωριστά νήματα απελευθερώνει το κύριο νήμα, επιτρέποντας την ομαλή λειτουργία και ενημέρωση του **UI**.
- **Ευκολία Συντήρησης**: Ο κώδικας έγινε πιο *μοδουλάρ*, με σαφή διαχωρισμό ανάμεσα στα δεδομένα του **UI** και σε αυτά που επεξεργάζεται ο *worker*.

11.6.6 Συμπεράσματα

Η αξιοποίηση *isolates* και *workers* αποδείχθηκε καθοριστικής σημασίας για την επιτυχή λειτουργία της εφαρμογής, επιτρέποντας την παράλληλη εκτέλεση υπολογιστικά απαιτητικών εργασιών χωρίς να διακυβεύεται η εμπειρία του χρήστη. Παρά τις πολυάριθμες προκλήσεις κατά τη διαδικασία μετάβασης, οι επιτευχθείσες βελτιώσεις όχι μόνο διασφάλισαν την ομαλή απόκριση της εφαρμογής, αλλά δημιούργησαν και μια στιβαρή βάση για μελλοντική ανάπτυξη και επέκταση. Η εμπειρία αυτή αναδεικνύει τη σημασία της προσαρμοστικότητας στον σχεδιασμό των αλγορίθμων και της εις βάθος κατανόησης των περιορισμών της εκάστοτε πλατφόρμας.

Κεφάλαιο 12ο: Αποτελέσματα

12.1 Εισαγωγή

Το παρόν κεφάλαιο συγκεντρώνει τα αποτελέσματα της εργασίας και εξετάζει τόσο την απόδοση του αλγορίθμου επίλυσης Εικονόσταυρων όσο και τη λειτουργικότητα της εφαρμογής από την πλευρά του χρήστη. Αρχικά, παρουσιάζεται η **Επισκόπηση δοκιμών και αξιολόγησης** (Ενότητα 2.2), όπου περιγράφονται αναλυτικά τα βήματα που ακολουθήθηκαν για τη δημιουργία και τη σταδιακή βελτίωση του αλγορίθμου. Ακολούθως, αναλύονται οι προκλήσεις που προέκυψαν κατά τη φάση ανάπτυξης, οι τεχνικές που εφαρμόστηκαν για την αντιμετώπισή τους, καθώς και τα μετρήσιμα οφέλη κάθε επιμέρους βελτίωσης.

Στη συνέχεια, γίνεται λόγος για την πρόοδο προς την πρώτη ολοκληρωμένη λύση, όπου παρουσιάζονται οι αρχικές μορφές του αλγορίθμου, οι λόγοι επιλογής συγκεκριμένων μεθόδων (όπως η «αριστερή» και «δεξιά» διαδικασία επίλυσης σε κάθε γραμμή ή στήλη) και τα πρώτα αποτελέσματα που επιτεύχθηκαν. Οι ενότητες αυτές φωτίζουν την εξελικτική πορεία του έργου, από μια στοιχειώδη υλοποίηση έως ένα πιο ολοκληρωμένο σύστημα.

Καταγράφονται συστηματικά οι *Βελτιώσεις* που υλοποιήθηκαν, όπως οι αλλαγές στην αρχιτεκτονική του αλγορίθμου, η αξιοποίηση της προσωρινής αποθήκευσης, η παράλληλη εκτέλεση (με απομονωμένες διεργασίες και εργαζόμενα νήματα), η βελτιστοποίηση των ελέγχων θέσεων, η αναδιάρθρωση της διεπαφής χρήστη και πολλές άλλες τεχνικές. Κάθε αλλαγή συνοδεύεται από μετρήσιμα δεδομένα (χρόνοι εκτέλεσης, αριθμός επαναλήψεων, χρήση μνήμης κ.λπ.), αποτυπώνοντας ξεκάθαρα τη συνεισφορά της κάθε στρατηγικής.

Παράλληλα, παρουσιάζεται η *Επίδειξη της λειτουργικότητας* της εφαρμογής, με λεπτομερή απεικόνιση των ροών χρήσης. Εκεί ο αναγνώστης μπορεί να διαπιστώσει πώς ένας χρήστης επιλέγει έναν έτοιμο Εικονόσταυρο για αυτόματη επίλυση ή πώς δημιουργεί έναν νέο Εικονόσταυρο, τον οποίο στη συνέχεια αναλαμβάνει να επιλύσει το σύστημα. Η συγκεκριμένη ενότητα εστιάζει στη διεπαφή, στους τρόπους αλληλεπίδρασης και στη συνολική εμπειρία που προσφέρει η εφαρμογή.

Έτσι, στο τελευταίο κεφάλαιο συνδυάζει την ποσοτική αποτίμηση της απόδοσης του αλγορίθμου με την ποιοτική επισκόπηση της διεπαφής και της χρηστικότητας, παρέχοντας μια ολοκληρωμένη εικόνα του τρόπου με τον οποίο οι σχεδιαστικές επιλογές και οι τεχνικές υλοποίησης οδηγούν σε ένα πλήρες σύστημα επίλυσης Εικονόσταυρων.

12.2 Επισκόπηση δοκιμών και αξιολόγηση

12.2.1 Πρόοδος προς την Πρώτη Ολοκληρωμένη Λύση

12.2.1.1 Εισαγωγή

Η διαδικασία ανάπτυξης του αλγορίθμου επίλυσης Εικονόσταυρων περιέλαβε πολλαπλά στάδια, ξεκινώντας από την αρχική αποτύπωση και ανάλυση των στοιχείων σε κάθε γραμμή και στήλη, μέχρι την τελική ολοκλήρωση του παζλ. Σε αυτό το τμήμα περιγράφεται βήμα προς βήμα η πορεία που ακολουθήθηκε, αναδεικνύοντας τις προκλήσεις που εμφανίστηκαν, τις αντίστοιχες λύσεις που εφαρμόστηκαν, καθώς και την επιστημονική βάση που οδήγησε στη δημιουργία ενός αποτελεσματικού και αξιόπιστου μηχανισμού επίλυσης.

12.2.1.2 Επίλυση Γραμμών

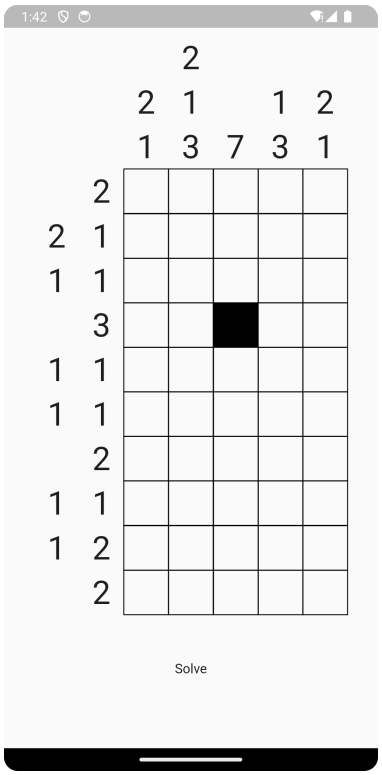
Η πρώτη φάση του αλγορίθμου αφορούσε την ανάλυση των γραμμών του Εικονόσταυρου, με στόχο τον εντοπισμό και την επιβεβαίωση των σίγουρα γεμάτων κελιών, βάσει των ενδείξεων που δίνονται για το παζλ. Η προσέγγιση

βασίστηκε σε δύο παράλληλες διαδικασίες: την «αριστερή» λύση (οι ενδείξεις τοποθετούνται από την αρχή της γραμμής) και τη «δεξιά» λύση (οι ενδείξεις τοποθετούνται από το τέλος). Τα σημεία όπου οι δύο λύσεις επικαλύπτονταν αναγνωρίζονταν ως σίγουρα γεμάτα κελιά.

Ο αλγόριθμος επεξεργαζόταν τη γραμμή ως μια συμβολοσειρά χαρακτήρων, χρησιμοποιώντας τη μέθοδο αντικατάστασης `replaceFirstMapped`. Η μέθοδος αυτή επιτρέπει την αντικατάσταση συγκεκριμένων τμημάτων μιας συμβολοσειράς με βάση ένα προκαθορισμένο μοτίβο. Για κάθε ένδειξη της γραμμής, το αρχικό μοτίβο αντικαθίστατο από μια ακολουθία γεμάτων κελιών («1») και κενών σημείων («?»), δημιουργώντας δύο ανεξάρτητες λύσεις: την «αριστερή» και τη «δεξιά». Αυτή η μέθοδος ήταν ιδιαίτερα αποδοτική, καθώς προσαρμοζόταν άμεσα στη συμβολοσειρά, χωρίς να χρειάζεται να εξεταστούν όλες οι πιθανές διατάξεις.

Καθώς η διαδικασία εξελισσόταν, οι δύο λύσεις συγκρίνονταν κελί προς κελί. Όπου υπήρχε ταύτιση, το κελί θεωρούνταν σίγουρα γεμάτο και ενημερωνόταν η κατάσταση της λύσης του παζλ, ενώ τα κενά σημεία παρέμεναν ανοιχτά για περαιτέρω ανάλυση σε επόμενα βήματα. Σε αυτή την πρώτη υλοποίηση, ο αλγόριθμος μπορούσε να χειριστεί μόνο γραμμές χωρίς ήδη συμπληρωμένα μέρη, ωστόσο η ταχύτητα εκτέλεσής του ήταν εντυπωσιακή για απλά μοτίβα, παρέχοντας μια ισχυρή βάση για τη μετέπειτα βελτίωση και γενίκευσή του.

Ένα χαρακτηριστικό παράδειγμα αφορά μια γραμμή πέντε κελιών με μία μόνο ένδειξη [3]. Στην «αριστερή» λύση, τα τρία πρώτα κελιά της γραμμής γεμίζουν με βάση την ένδειξη, ενώ στην «δεξιά» γεμίζουν τα τρία τελευταία κελιά. Η επικάλυψη των δύο λύσεων οδηγεί στη σίγουρη αναγνώριση του κεντρικού κελιού ως γεμάτου. Αυτή η απλή, αλλά αποτελεσματική προσέγγιση αποτέλεσε το θεμέλιο για την περαιτέρω ανάπτυξη του αλγορίθμου.



Σχήμα 12.1: Αποτέλεσμα λύτη με εφαρμογή στις γραμμές

Τέλος, οι εικόνες που συνοδεύουν αυτή τη φάση απεικονίζουν γραμμές όπου μόνο ένα τμήμα έχει ολοκληρωθεί. Για παράδειγμα, μια γραμμή με πέντε κελιά και ένδειξη [3] εμφανίζεται με το μεσαίο τμήμα της γεμάτο, επιβεβαιώνοντας την επιτυχία του αλγορίθμου στην αναγνώριση των σίγουρα γεμάτων κελιών.

12.2.1.3 Επίλυση Στηλών

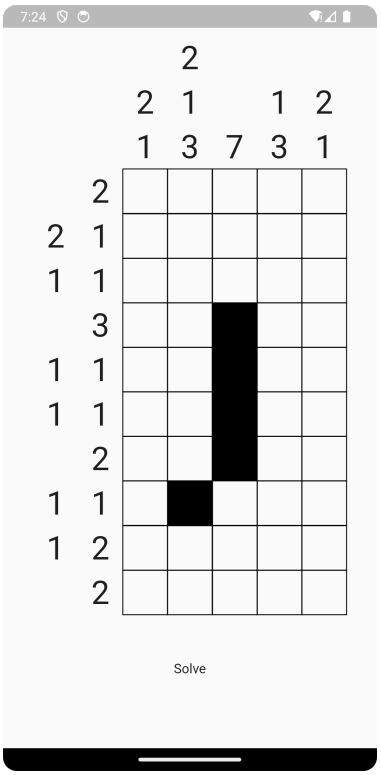
Μετά την επιτυχημένη εφαρμογή της λογικής για την ανάλυση γραμμών, η επόμενη φάση επεκτάθηκε στις στήλες. Οι στήλες αντιμετωπίστηκαν ως ανεξάρτητες συμβολοσειρές, και εφαρμόστηκε η ίδια λογική της ”αριστερής” και ”δεξιάς” λύσης, προσαρμοσμένη στις ενδείξεις της κάθε στήλης.

Για κάθε στήλη, η ενεργή λύση μετασχηματιζόταν σε συμβολοσειρά και αναλύονταν οι ενδείξεις της. Οι ενδείξεις τοποθετούνταν από την αρχή της στήλης (”αριστερή” λύση) και από το τέλος (”δεξιά” λύση), όπως είχε γίνει και για τις γραμμές. Η μέθοδος `replaceFirstMapped` χρησιμοποιήθηκε και εδώ για να αντικαταστήσει συγκεκριμένα μοτίβα χαρακτήρων. Το αποτέλεσμα ήταν η άμεση δημιουργία λύσεων, χωρίς την ανάγκη εξέτασης όλων των πιθανών διατάξεων, γεγονός που καθιστούσε τη διαδικασία εξαιρετικά αποδοτική.

Ο αλγόριθμος τοποθετούσε τα γράμματα των πλευρικών λύσεων απευθείας στη συμβολοσειρά της στήλης. Αυτό συνέβαλε στη δραστική μείωση του χρόνου επεξεργασίας, ειδικά για στήλες με απλά μοτίβα ενδείξεων, και επέτρεψε την ταχύτερη σύγκριση των αριστερών και δεξιών λύσεων για τον εντοπισμό των σίγουρα γεμάτων κελιών.

Ένα χαρακτηριστικό αποτέλεσμα αυτής της φάσης ήταν η δυνατότητα ολοκλήρωσης τριών από τις πέντε στήλες σε ένα παράδειγμα παζλ. Οι στήλες επεξεργάστηκαν ανεξάρτητα από την προηγούμενη ανάλυση των γραμμών, καθώς η συνδυαστική λειτουργία του αλγορίθμου δεν είχε ακόμη εισαχθεί. Τα δεδομένα από τις γραμμές αξιοποιήθηκαν για τη βελτίωση της ακρίβειας, παρέχοντας ένα δυναμικό σύστημα αλληλεπίδρασης μεταξύ γραμμών και στηλών.

Η συγκεκριμένη φάση αποτέλεσε σημαντικό βήμα για την προσαρμογή του αλγορίθμου σε πιο σύνθετα μοτίβα, εδραιώνοντας την αποτελεσματικότητά του για μεγαλύτερα και πιο απαιτητικά παζλ.

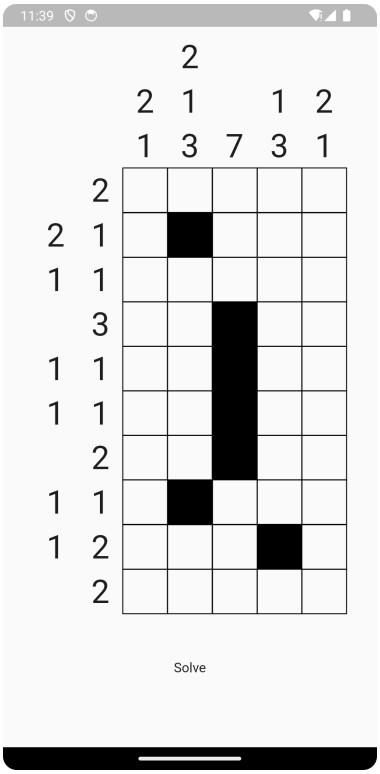


Σχήμα 12.2: Αποτέλεσμα λύτη με εφαρμογή στις στήλες

12.2.1.4 Συνδυαστική Επίλυση Γραμμών και Στηλών

Μετά την ανεξάρτητη ανάλυση γραμμών και στηλών, η επόμενη φάση περιλάμβανε την ενοποίηση των δύο κωδίκων σε μία ενιαία εκτέλεση. Ο αλγόριθμος ρύθμισε κάθε διαδικασία ώστε να εκτελεί μία φορά την επίλυση όλων των γραμμών και στη συνέχεια όλων των στηλών, χωρίς τα δεδομένα που συμπληρώθηκαν κατά την ανάλυση μιας γραμμής να επηρεάζουν την επίλυση της επόμενης γραμμής.

Η διαδικασία αυτή στόχευε στη δημιουργία μιας πιο ολοκληρωμένης εικόνας της λύσης, όπου τα αποτελέσματα κάθε εκτέλεσης γίνονταν μέρος της συνολικής προόδου. Σε αντίθεση με τις προηγούμενες φάσεις, όπου κάθε μέθοδος λειτουργούσε απομονωμένα, η συνδυαστική προσέγγιση επέτρεψε την επίλυση των παζλ με τρόπο που προσομοίαζε τη συνολική πρόοδο, χωρίς να υπάρχει αλληλεπίδραση μεταξύ των γραμμών και στηλών κατά τη διάρκεια της εκτέλεσης.



Σχήμα 12.3: Αποτέλεσμα λύτη με εφαρμογή και στις γραμμές και στις στήλες

12.2.1.5 Διαγραφή κουτιών

Η προσθήκη ενός μηχανισμού για τη διαγραφή κελιών που δεν μπορούσαν να γεμίσουν αποτέλεσε ένα σημαντικό βήμα για τη βελτίωση της αποτελεσματικότητας του αλγορίθμου. Αυτή η διαδικασία σχεδιάστηκε για να αναγνωρίζει τα σίγουρα κενά κελιά, τα οποία δεν μπορούσαν να ανήκουν σε καμία ένδειξη, και να τα αποκλείει από τη λύση.

Ο αλγόριθμος λειτουργούσε εξετάζοντας περιοχές γύρω από ήδη γεμάτα κελιά και συγκρίνοντας τα αποτελέσματα των ”αριστερών” και ”δεξιών” λύσεων. Για κάθε κελί, ελέγχονταν τα πιθανά αποτελέσματα από τις πλευρικές λύσεις. Αν οι δύο λύσεις ταυτίζονταν και έδειχναν ότι ένα κελί δεν μπορεί να ανήκει σε καμία ένδειξη, τότε το κελί αυτό διαγραφόταν και σημειωνόταν με το σύμβολο ”X”.

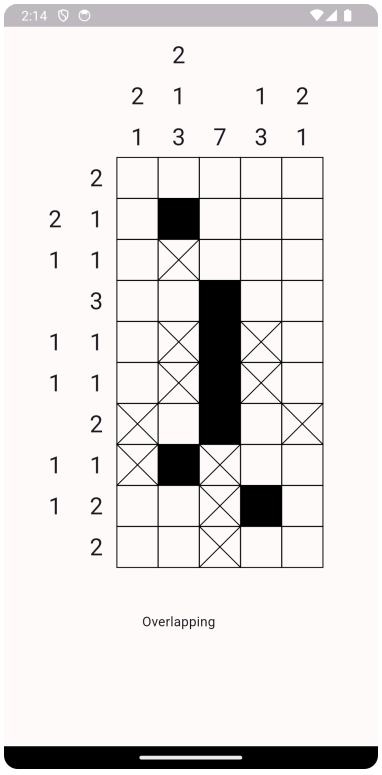
Η διαδικασία αυτή περιλάμβανε δύο βασικά στάδια: πρώτον, την ανάλυση των ”χαρακτηριστικών περιοχών” πριν και μετά από κάθε ένδειξη για να αξιολογηθεί η συμβατότητα των κελιών, και δεύτερον, τη σήμανση των σίγουρα

κενών κελιών με βάση τη συμφωνία των πλευρικών λύσεων. Αυτή η σύγκριση των αποτελεσμάτων επέτρεψε στον αλγόριθμο να επικυρώσει τις διαγραφές με ακρίβεια, μειώνοντας σημαντικά τον αριθμό των λανθασμένων διαγραφών.

Για παράδειγμα, αν ένα κελί βρίσκεται δίπλα σε μια γεμάτη ομάδα και οι πλευρικές λύσεις συμφωνούν ότι δεν μπορεί να χωρέσει άλλη ένδειξη, τότε το κελί αυτό χαρακτηρίζεται ως κενό. Επιπλέον, αν υπάρχουν κενά που περιβάλλονται από ήδη συμπληρωμένες ενδείξεις και δεν υπάρχει αρκετός χώρος για την τοποθέτηση άλλων ομάδων, τότε αυτά τα κενά αποκλείονται επίσης.

Αυτή η μέθοδος όχι μόνο βελτίωσε την αποδοτικότητα του αλγορίθμου, αλλά παρείχε επίσης μια πιο καθαρή εικόνα του παζλ για τα επόμενα στάδια ανάλυσης. Οι εικόνες που συνοδεύουν αυτή τη φάση δείχνουν πώς τα ”X” τοποθετήθηκαν στα κατάλληλα κελιά, επιτρέποντας την καλύτερη κατανόηση της προόδου του αλγορίθμου.

Η εισαγωγή της διαγραφής κενών κελιών ήταν καθοριστική για την αντιμετώπιση μεγαλύτερων και πιο περίπλοκων παζλ, καθώς μείωσε σημαντικά τον αριθμό των πιθανών λύσεων που έπρεπε να εξεταστούν, εξασφαλίζοντας παράλληλα την ακρίβεια των αποτελεσμάτων.



Σχήμα 12.4: Αποτέλεσμα λύτη με διαγραφή κελιών

12.2.1.6 Λύση παζλ με επαναποστολή νέας λύσης προς λύση

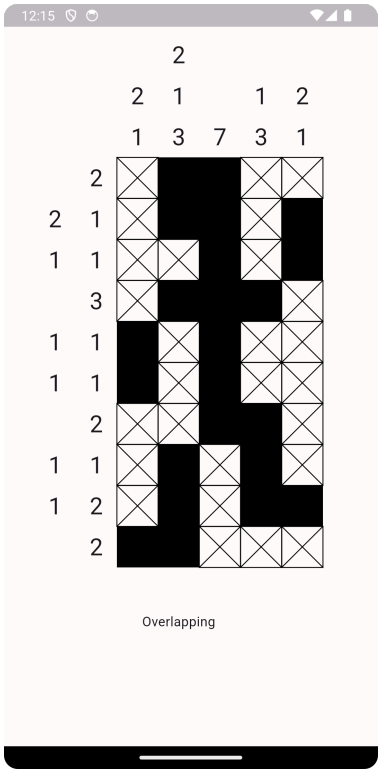
Σε αυτή τη φάση της ανάπτυξης, ο αλγόριθμος δεν είχε ακόμα τη δυνατότητα να εντοπίζει και να προωθεί προς επίλυση μόνο τις γραμμές ή τις στήλες που απαιτούσαν περαιτέρω ανάλυση. Αντίθετα, η εφαρμογή εκτελούσε μία πλήρη ανάλυση όλων των γραμμών και στη συνέχεια όλων των στηλών σε κάθε εκτέλεση. Μόλις ολοκληρωνόταν αυτή η διαδικασία, ο αλγόριθμος σταματούσε, ανεξάρτητα από το αν υπήρχαν περισσότερα δεδομένα που θα μπορούσαν να οδηγήσουν σε νέα συμπεράσματα.

Ο μηχανισμός που επέτρεπε την πρόοδο στη λύση του παζλ βασιζόταν στη χειροκίνητη επανεκτέλεση του αλγορίθμου μέσω του κουμπιού **”SOLVE”**. Κάθε φορά που ο χρήστης πάταγε το κουμπί, ο αλγόριθμος έτρεχε ξανά,

χρησιμοποιώντας τα νέα στοιχεία που είχαν συμπληρωθεί από την προηγούμενη εκτέλεση. Αυτή η διαδικασία, αν και δεν ήταν πλήρως αυτοματοποιημένη, εξασφάλιζε συνεχή πρόοδο στην επίλυση του παζλ.

Με κάθε επανάληψη, τα γεμάτα κελιά από τις προηγούμενες εκτελέσεις συνέβαλαν στη βελτίωση της ακρίβειας της λύσης, οδηγώντας σε σταδιακή μείωση της αβεβαιότητας και ολοκλήρωση του παζλ. Ο αλγόριθμος κατάφερε έτσι να επιλύει ακόμη και πιο σύνθετα παζλ, αν και απαιτούσε πολλαπλές εκτελέσεις για να φτάσει στην τελική λύση.

Αυτή η προσέγγιση, αν και προσωρινή, ανέδειξε την αποτελεσματικότητα του αλγορίθμου και έθεσε τη βάση για τη μελλοντική ανάπτυξη μιας αυτοματοποιημένης διαδικασίας που θα μπορούσε να εντοπίζει και να επανεκκινεί αυτόματα τις απαραίτητες γραμμές ή στήλες, βελτιώνοντας σημαντικά τη συνολική απόδοση.



Σχήμα 12.5: Αποτέλεσμα λύτη μετά από επαναλαμβανόμενη εφαρμογή λύτη

12.2.2 Βελτιώσεις

12.2.2.1 Εισαγωγή

Η ανάπτυξη του αλγορίθμου επίλυσης Εικονόσταυρων ήταν μια δυναμική διαδικασία που δεν περιορίστηκε στην αρχική του υλοποίηση, αλλά συνεχώς εξελισσόταν μέσα από συστηματικές βελτιώσεις. Οι αλλαγές αυτές δεν στόχευαν μόνο στη μείωση του χρόνου επίλυσης αλλά και στην ενίσχυση της απόδοσης, της χρηστικότητας και της εμπειρίας του χρήστη. Κατά τη διάρκεια αυτής της διαδρομής, προσεγγίστηκαν ποικίλα ζητήματα, όπως η αφαίρεση περιττών λειτουργιών, η χρήση τεχνικών προσωρινής αποθήκευσης, η εισαγωγή παράλληλης επεξεργασίας και η ανασχεδίαση του τρόπου παρουσίασης και λειτουργίας της διεπαφής χρήστη.

Οι βελτιώσεις που υλοποιήθηκαν δεν επικεντρώθηκαν μόνο στον πυρήνα του αλγορίθμου αλλά επεκτάθηκαν και στο περιβάλλον της εφαρμογής. Η ανασχεδίαση του πλέγματος και οι βελτιώσεις στη ροή των σελίδων εξάλειψαν προβλήματα καθυστέρησης, ενώ η διαδικασία αναδιοργάνωσης και τεκμηρίωσης του κώδικα εξασφάλισε ότι το έργο έγινε όχι μόνο πιο αποδοτικό, αλλά και πιο κατανοητό και ευανάγνωστο, τόσο για μελλοντικούς συνεργάτες

όσο και για εμένα την ίδια.

Αυτή η ενότητα εξετάζει τις επιμέρους βελτιώσεις που υλοποιήθηκαν, εξηγεί τη λογική πίσω από την εφαρμογή τους και παρουσιάζει την επίδρασή τους στην απόδοση του αλγορίθμου. Μέσα από αυτή τη διαδικασία, ο αλγόριθμος εξελίχθηκε σε ένα εργαλείο που όχι μόνο μπορεί να επιλύει σύνθετα παζλ, αλλά και να το κάνει με έναν τρόπο που είναι αποδοτικός, ευχάριστος και αποτελεσματικός για τον χρήστη.

12.2.2.2 Αφαίρεση εκτυπώσεων κοσνόλας (*print*)

Η αφαίρεση των `print` statements αποτέλεσε την πρώτη και πιο απλή βελτίωση του αλγορίθμου. Παρότι χρησίμευαν για λόγους debugging, η χρήση τους προσέθετε σημαντικό υπολογιστικό χρόνο κατά την εκτέλεση του αλγορίθμου. Μετά την αφαίρεση των `print`, ο χρόνος εκτέλεσης μειώθηκε δραστικά, ιδιαίτερα σε μικρά παζλ όπου η εκτέλεση επιταχύνθηκε κατά 1 έως 1.5 δευτερόλεπτα. Αντίστοιχα, στα μεγαλύτερα παζλ, η απουσία των `print` οδήγησε σε αισθητή μείωση του χρόνου επίλυσης, κάνοντας τον αλγόριθμο πιο αποδοτικό.

12.2.2.3 Μείωση επαναλήψεων με εκκίνηση και τερματισμό ελέγχου σε “?”

Η εισαγωγή της μεθόδου περιορισμού των επαναλήψεων βασίστηκε στην παρατήρηση ότι οι έλεγχοι του αλγορίθμου μπορούσαν να περιοριστούν στις περιοχές όπου υπήρχαν κενά (”?”) στη λύση. Αντί να ελέγχει όλα τα στοιχεία, ο αλγόριθμος επικεντρώθηκε μόνο στα σημεία ενδιαφέροντος, μειώνοντας έτσι σημαντικά τον αριθμό των επαναλήψεων. Αυτή η βελτίωση μείωσε τον χρόνο εκτέλεσης κατά περίπου 40 δευτερόλεπτα στο μεγαλύτερο παζλ, ενώ βελτίωσε τη συνολική αποδοτικότητα του αλγορίθμου.

12.2.2.4 Προσπάθεια μείωσης ελέγχων γραμμών

Η μεθοδολογία για τη μείωση των ελέγχων γραμμών περιλάμβανε τη χρήση ενός φίλτρου που απέκλειε γραμμές από την επεξεργασία εάν δεν πληρούσαν συγκεκριμένα κριτήρια. Πιο συγκεκριμένα, γραμμές όπου το άθροισμα των στοιχείων τους δεν έφτανε τουλάχιστον το 1/4 του πλάτους του παζλ και το άθροισμα των ήδη συμπληρωμένων κελιών τους δεν πληρούσε αυτή την προϋπόθεση, εξαιρούνταν από την περαιτέρω ανάλυση. Ωστόσο, η μέθοδος αυτή αφαιρέθηκε, καθώς δεν υπήρχε σαφής μαθηματική, λογική ή πρακτική απόδειξη ότι δεν επηρέαζε αρνητικά την επίλυση του παζλ. Υπήρχε ο κίνδυνος, υπό συγκεκριμένες συνθήκες, να αποκλείσει γραμμές που θα μπορούσαν να συμβάλουν καθοριστικά στη λύση, κάτι που καθιστούσε τη χρήση της αναξιόπιστη.

12.2.2.5 Ομαδοποίηση συμπλήρωσης κουτιών

Αρχικά, η τεχνική της ομαδοποίησης εφαρμόστηκε στη διαδικασία διαγραφής κουτιών (*cross out*), αντικαθιστώντας τις επαναλήψεις με τη χρήση `regex`. Με αυτόν τον τρόπο, ο αλγόριθμος μπορούσε να εντοπίζει τα σημεία που χρειάζονταν διαγραφή και να τα ομαδοποιεί, εκτελώντας τη διαδικασία ως ένα βήμα αντί για πολλές ξεχωριστές λειτουργίες. Παρόλο που η διαφορά στον χρόνο εκτέλεσης ήταν οριακά μικρή, η προσέγγιση αυτή μείωσε την πολυπλοκότητα του κώδικα και διευκόλυνε τη διαχείριση των δεδομένων.

Αργότερα, λόγω των θετικών αποτελεσμάτων, η ίδια μεθοδολογία εφαρμόστηκε και στη συμπλήρωση κουτιών με στοιχεία. Συγκεκριμένα, αντί κάθε νέο κουτάκι που συμπληρωνόταν να εμφανίζεται ως ξεχωριστό βήμα, ομαδοποιήθηκαν τα κουτάκια που προέρχονταν από το ίδιο στοιχείο και προστέθηκαν μαζί. Αυτό είχε ως αποτέλεσμα να εμφανίζεται η διαδικασία ως ένα ενιαίο βήμα, μειώνοντας τον αριθμό των φορών που ο κώδικας έπρεπε να τρέχει για την προσθήκη δεδομένων στη μνήμη. Παράλληλα, περιορίστηκε η συνολική χρήση μνήμης, ενώ η εμπειρία του χρήστη βελτιώθηκε σημαντικά. Πλέον, οι πληροφορίες που εμφανίζονταν ήταν πιο στοχευμένες και οργανωμένες, προσφέροντας μια σαφέστερη εικόνα της προόδου της επίλυσης του παζλ.

12.2.2.6 Προσθήκη προσωρινής αποθήκευσης (*Caching*)

Η προσθήκη *caching* αποτέλεσε μια από τις πιο αποτελεσματικές και καίριες βελτιώσεις στον αλγορίθμο επίλυσης. Η υλοποίηση περιλάμβανε τη δημιουργία μιας δομής `Map<String, bool>`, όπου το κλειδί (`key`) αποτελούνταν από μια συμβολοσειρά που συνδύαζε πληροφορίες σχετικά με τις ενδείξεις (*clues*), τη θέση τους (*clue index*), τη λύση (*solution*) και τη θέση στη λύση (*solution index*), διαχωρισμένα με κόμμα. Η τιμή (`value`) του χάρτη ήταν ένα boolean που υποδείκνυε αν μια συγκεκριμένη ένδειξη μπορούσε να τοποθετηθεί σε μια συγκεκριμένη θέση της λύσης.

Αυτή η τεχνική είχε άμεσα και εντυπωσιακά αποτελέσματα, μειώνοντας σημαντικά τον χρόνο εκτέλεσης. Στο παζλ *"Probably Not"*, ο χρόνος επίλυσης μειώθηκε από 1 λεπτό και 20 δευτερόλεπτα σε μόλις 17 δευτερόλεπτα, σημειώνοντας μια μείωση κατά 78%. Στο μεγαλύτερο παζλ της λίστας, το *"Swing"*, η μείωση του χρόνου ήταν εξίσου εντυπωσιακή, με τον χρόνο εκτέλεσης να πέφτει από 8.3 λεπτά σε 7.2 λεπτά. Αυτές οι βελτιώσεις κατέστησαν τον αλγόριθμο πιο αποδοτικό και ικανό να χειριστεί πιο απαιτητικά προβλήματα.

Η προσέγγιση του *caching* είχε επίσης θετική επίδραση στη χρήση μνήμης και στον αριθμό των επαναλήψεων που χρειαζόταν να εκτελεστούν. Παρόλα αυτά, έγινε προσπάθεια να προστεθεί και μηχανισμός καθαρισμού του *cache* για περιπτώσεις όπου η χρήση μνήμης θα μπορούσε να γίνει υπερβολική. Ωστόσο, τα παζλ δεν ήταν αρκετά μεγάλα ώστε να προκύψει ανάγκη για καθαρισμό, και συνεπώς αυτή η λειτουργία δεν ενεργοποιήθηκε στη συγκεκριμένη φάση.

Επιπλέον, η τεχνική του *caching* αρχικά δοκιμάστηκε και σε άλλες περιοχές του αλγορίθμου, όπως στον έλεγχο των πλευρικών στοιχείων (*side clues*) κάθε κελιού. Ενώ σε μικρότερα παζλ αυτός ο επιπλέον έλεγχος κατάφερε να μειώσει περαιτέρω τις επαναλήψεις, στα μεγαλύτερα παζλ πρόσθεσε χρόνο εκτέλεσης, φτάνοντας σε ορισμένες περιπτώσεις να αυξάνει τον συνολικό χρόνο κατά 10 λεπτά. Λόγω αυτής της παρατήρησης, ο έλεγχος αυτός αφαιρέθηκε για να διασφαλιστεί η συνολική αποδοτικότητα του αλγορίθμου.

Η εφαρμογή του *caching* αποτέλεσε ουσιαστική αλλαγή στον τρόπο λειτουργίας του αλγορίθμου, προσφέροντας μια από τις μεγαλύτερες βελτιώσεις στον χρόνο και την ακρίβεια της επίλυσης. Αυτό το βήμα δεν ήταν μόνο τεχνικά επιτυχημένο, αλλά και καθοριστικό για την αντιμετώπιση των περιορισμών σε πιο σύνθετα παζλ, ανοίγοντας το δρόμο για περαιτέρω ανάπτυξη και εφαρμογή της ίδιας λογικής σε άλλες περιοχές του αλγορίθμου.

12.2.2.7 Ακόμα καλύτερη μείωση ελέγχων θέσεων

Η βελτίωση των ελέγχων θέσεων αποτέλεσε ένα σημαντικό σημείο καμπής για τη συνολική απόδοση του αλγορίθμου. Στόχος ήταν η δραστική μείωση των επαναλήψεων που εκτελούνταν κατά την ανάλυση μιας γραμμής ή στήλης, επικεντρώνοντας την προσοχή του αλγορίθμου μόνο στα τμήματα της λύσης όπου υπήρχε πραγματική πιθανότητα να τοποθετηθούν στοιχεία.

Μέχρι εκείνο το σημείο, η μόνη βελτίωση που είχε εφαρμοστεί για τη μείωση των επαναλήψεων ήταν η τεχνική που περιγράφεται στην ενότητα "2.2.2.3 Μείωση επαναλήψεων με εκκίνηση και τερματισμό ελέγχου σε '?'". Αυτή η βελτίωση περιόριζε τους ελέγχους μόνο στις περιοχές όπου υπήρχαν κενά ("?"). Παρόλο που η τεχνική αυτή είχε μειώσει τον αριθμό των επαναλήψεων και βελτίωσε τον χρόνο εκτέλεσης, η νέα προσέγγιση που εφαρμόστηκε με τη δυναμική προσαρμογή των ελάχιστων και μέγιστων σημείων ελέγχου αποδείχθηκε ακόμα πιο αποδοτική. Εστιάζοντας μόνο στις περιοχές όπου υπήρχε πραγματική πιθανότητα να τοποθετηθούν στοιχεία, η νέα μέθοδος μείωσε ακόμη περισσότερο τον αριθμό των περιττών υπολογισμών, οδηγώντας σε εντυπωσιακή μείωση του χρόνου επίλυσης.

Η βελτίωση βασίστηκε στην εισαγωγή δυναμικών παραμέτρων που καθόριζαν την ελάχιστη και μέγιστη θέση όπου κάθε στοιχείο μιας γραμμής ή στήλης μπορούσε να εξεταστεί. Οι παράμετροι αυτές, γνωστές ως `minStartingPoint`

και `maxEndingPoint`, υπολογίζονταν για κάθε στοιχείο ξεχωριστά, λαμβάνοντας υπόψη το μέγεθος και τις θέσεις των υπόλοιπων στοιχείων στη γραμμή ή στήλη.

Η παράμετρος `minStartingPoint` καθόριζε την πρώτη θέση που μπορούσε να εξεταστεί για ένα στοιχείο. Για τον υπολογισμό της, ο αλγόριθμος λάμβανε υπόψη το άθροισμα των μηκών των προηγούμενων στοιχείων και των απαραίτητων κενών μεταξύ τους. Με αυτόν τον τρόπο, αγνοούσε τις αρχικές θέσεις που ήταν αδύνατο να φιλοξενήσουν το στοιχείο, εξαλείφοντας μεγάλο αριθμό περιττών ελέγχων.

Αντίστοιχα, η παράμετρος `maxEndingPoint` καθόριζε την τελευταία θέση που μπορούσε να εξεταστεί για ένα στοιχείο. Ο αλγόριθμος υπολόγιζε αυτήν την τιμή αφαιρώντας το άθροισμα των μηκών των επόμενων στοιχείων και των απαραίτητων κενών, καθώς και την τιμή του ελεγχόμενου στοιχείου, από το συνολικό μήκος της γραμμής ή στήλης. Με αυτόν τον τρόπο, απέκλειε από τους ελέγχους τις θέσεις που βρίσκονταν πέρα από την πιθανή τελική θέση του στοιχείου.

Η εισαγωγή αυτών των παραμέτρων μείωσε σημαντικά τον αριθμό των ελέγχων που εκτελούνταν, καθώς ο αλγόριθμος περιοριζόταν σε ένα μικρότερο, πιο στοχευμένο εύρος θέσεων για κάθε στοιχείο. Αυτό είχε ως αποτέλεσμα τη μείωση της πολυπλοκότητας της διαδικασίας και τη δραματική βελτίωση της αποδοτικότητας του αλγορίθμου, ιδιαίτερα σε μεγαλύτερα και πιο απαιτητικά παζλ.

12.2.2.8 Χρήση Isolates και Workers

Η εισαγωγή των *Isolates* και *Workers* αποτέλεσε μία από τις πιο θεαματικές βελτιώσεις του αλγορίθμου επίλυσης Εικονόσταυρων, επιφέροντας σημαντική αύξηση στην αποδοτικότητα και παράλληλα βελτίωση της εμπειρίας του χρήστη. Πριν από αυτήν τη βελτίωση, ο αλγόριθμος εκτελούνταν αποκλειστικά στο κύριο νήμα, κάτι που προκαλούσε σημαντικά προβλήματα, ιδιαίτερα σε μεγαλύτερα παζλ. Ο χρόνος επίλυσης ήταν υψηλός και, κατά τη διάρκεια της επεξεργασίας, η διεπαφή χρήστη (**UI**) πάγωνε, κάνοντας την εφαρμογή να φαίνεται μη αποκρίσιμη.

Η χρήση των *Isolates* επέτρεψε την εκτέλεση του αλγορίθμου σε ένα απομονωμένο περιβάλλον, ανεξάρτητο από το κύριο νήμα. Αυτό σήμαινε ότι ο υπολογισμός των δεδομένων μπορούσε να συνεχιστεί χωρίς να επηρεάζει τη λειτουργικότητα της διεπαφής χρήστη. Παράλληλα, μέσω των *Workers*, διασφαλίστηκε η αποστολή των αποτελεσμάτων και της προόδου από το απομονωμένο νήμα προς το κύριο, επιτρέποντας την ενημέρωση του **UI** σε πραγματικό χρόνο. Ο χρήστης μπορούσε πλέον να βλέπει το παζλ να συμπληρώνεται σταδιακά, χωρίς διακοπές ή καθυστερήσεις, γεγονός που ενίσχυσε την αίσθηση της αλληλεπίδρασης και της διαφάνειας.

Τα αποτελέσματα αυτής της βελτίωσης ήταν εντυπωσιακά. Σε ένα από τα μεγαλύτερα παζλ της λίστας, ο χρόνος επίλυσης μειώθηκε στα 19 δευτερόλεπτα, μία μείωση της τάξης του 80% σε σχέση με την προηγούμενη εκδοχή του αλγορίθμου. Εκτός από τη σημαντική βελτίωση της ταχύτητας, η δυνατότητα παρακολούθησης της επίλυσης σε πραγματικό χρόνο πρόσθεσε μια νέα διάσταση στη λειτουργικότητα της εφαρμογής, καθιστώντας τη διαδικασία πιο εντυπωσιακή και ευχάριστη για τον χρήστη. Η προσθήκη των *Isolates* και *Workers* όχι μόνο έλυσε το πρόβλημα της αποδοτικότητας, αλλά έθεσε και τις βάσεις για μελλοντική επεκτασιμότητα και προσαρμογή του αλγορίθμου σε ακόμα πιο απαιτητικά παζλ.

12.2.2.9 Εξυπνότερη ταξινόμηση της αρχικής στοίβας γραμμών και στηλών

Μια από τις πιο καθοριστικές βελτιώσεις για τη βελτίωση της αποδοτικότητας του αλγορίθμου ήταν η αναβάθμιση της διαδικασίας ταξινόμησης της αρχικής στοίβας γραμμών και στηλών. Στην αρχική υλοποίηση, η στοίβα χτιζόταν απλά με όλες τις γραμμές και στη συνέχεια όλες τις στήλες, χωρίς να λαμβάνεται υπόψη η σχετική σημασία ή προτεραιότητα κάθε γραμμής ή στήλης. Αυτή η προσέγγιση, αν και λειτουργική, οδηγούσε σε περιττούς υπολογισμούς, καθώς ο αλγόριθμος εξέταζε περιοχές του παζλ που ήταν λιγότερο πιθανό να περιέχουν χρήσιμες

πληροφορίες.

Η αναβάθμιση στηρίχθηκε στην παρατήρηση ότι οι γραμμές και οι στήλες με μεγαλύτερο άθροισμα στοιχείων είχαν περισσότερες πιθανότητες να οδηγήσουν σε συμπληρωμένα κελιά. Για αυτό, αποφασίστηκε να εισαχθεί μια στρατηγική ταξινόμησης που κατέτασσε γραμμές και στήλες στη στοίβα με βάση το συνολικό άθροισμα των στοιχείων τους. Όσο μεγαλύτερο ήταν το άθροισμα, τόσο νωρίτερα τοποθετούνταν στη στοίβα προς επεξεργασία. Η λογική πίσω από αυτή την επιλογή ήταν ότι γραμμές ή στήλες με περισσότερα στοιχεία παρέχουν περισσότερους περιορισμούς και έτσι είναι πιθανότερο να αποδώσουν χρήσιμα δεδομένα κατά την επεξεργασία τους.

Παρατηρήθηκε επίσης ότι πολλές από τις πρώτες γραμμές που εκτελούνταν πριν την εφαρμογή της στρατηγικής ταξινόμησης είχαν χαμηλό άθροισμα στοιχείων. Αυτές οι γραμμές, συχνά με λίγα ή καθόλου συμπληρωμένα κελιά, απαιτούσαν την εκτέλεση περιττών ελέγχων, ειδικά όταν δεν υπήρχαν αρκετά δεδομένα που να περιορίζουν τις πιθανές διατάξεις των στοιχείων. Με τη νέα ταξινόμηση, αυξήθηκαν οι πιθανότητες να επεξεργαστούν πρώτες οι γραμμές και οι στήλες με τουλάχιστον ένα συμπληρωμένο στοιχείο, επιτρέποντας στον αλγόριθμο να επικεντρωθεί σε περιοχές του παζλ με μεγαλύτερη σημασία.

Επιπλέον, η νέα στρατηγική βελτίωσε και τη σειρά με την οποία γίνονταν οι εναλλαγές μεταξύ γραμμών και στηλών. Στην αρχική έκδοση, ο αλγόριθμος εκτελούσε πρώτα όλες τις γραμμές και στη συνέχεια όλες τις στήλες, κάτι που οδηγούσε σε καθυστερήσεις μέχρι να επεξεργαστεί τις επηρεασμένες γραμμές ή στήλες. Με τη μίξη της εκτέλεσης γραμμών και στηλών, δημιουργήθηκε μια πιο αποδοτική ροή, όπου οι αλλαγές που προκαλούνταν από την ανάλυση μιας γραμμής μπορούσαν να επηρεάσουν άμεσα τη λύση στις αντίστοιχες στήλες, και αντίστροφα. Έτσι, η πρόοδος του αλγορίθμου έγινε πιο ομαλή και πιο γρήγορη.

Η αλλαγή αυτή επηρέασε τόσο τον αριθμό των επαναλήψεων όσο και τον συνολικό χρόνο εκτέλεσης. Ένας χαρακτηριστικός λόγος για τη βελτίωση ήταν ότι οι γραμμές ή οι στήλες με μεγάλα αθροίσματα στοιχείων είχαν αυξημένες πιθανότητες να περιέχουν δεδομένα που βοηθούσαν στην ταχύτερη συμπλήρωση του παζλ. Για παράδειγμα, γραμμές ή στήλες με ένα μόνο στοιχείο εξετάζονταν πλέον όταν υπήρχαν ήδη κάποια γεμάτα κελιά, μειώνοντας έτσι τον αριθμό των περιττών ελέγχων.

Αξίζει να σημειωθεί ότι δοκιμάστηκε και η αντίστροφη ταξινόμηση, από το μικρότερο στο μεγαλύτερο άθροισμα. Παρόλο που αυτή η στρατηγική αύξησε τον αριθμό των βημάτων του αλγορίθμου, ο συνολικός χρόνος εκτέλεσης και ο αριθμός των επαναλήψεων που εκτελέστηκαν ήταν μικρότερος σε σύγκριση με τη μη ταξινομημένη στοίβα. Ωστόσο, η ταξινόμηση από το μεγαλύτερο στο μικρότερο άθροισμα αποδείχθηκε πιο αποτελεσματική, με ακόμα λιγότερες επαναλήψεις και μικρότερο συνολικό χρόνο.

Συνολικά, η έξυπνη ταξινόμηση της αρχικής στοίβας βελτίωσε θεαματικά την αποδοτικότητα του αλγορίθμου, μειώνοντας τις περιττές επαναλήψεις και τον συνολικό χρόνο επίλυσης. Παράλληλα, δημιούργησε τις βάσεις για περαιτέρω βελτιώσεις, καθώς αποδείχθηκε ότι η προτεραιοποίηση των δεδομένων μπορεί να παίξει καθοριστικό ρόλο στη βελτίωση της αποδοτικότητας των αλγορίθμων επίλυσης.

12.2.2.10 Βελτίωση Ροής Σελίδων και Απόδοσης Κατά τη Μετάβαση

Η ροή της εφαρμογής, συμπεριλαμβανομένου του χρόνου φόρτωσης, της μετάβασης μεταξύ σελίδων και της κύλισης στη σελίδα, παρουσίαζε σημαντικές υστερήσεις στην αρχική υλοποίηση. Το πρόβλημα ήταν ιδιαίτερα εμφανές κατά την κύλιση σε σελίδες με μεγάλες λίστες ή κατά τη δημιουργία και επεξεργασία ενός παζλ, όπου η απόδοση ήταν αισθητά χαμηλότερη, με το UI να κολλάει ή να καθυστερεί. Η κύρια αιτία αυτού του προβλήματος ήταν η χρήση ενός `GridView`, το οποίο είχε το μέγεθος του παζλ και περιλάμβανε ως *items* ξεχωριστά widgets για κάθε κελί του πλέγματος.

Κάθε στοιχείο του `GridView` περιλάμβανε έναν *CustomPainter* για την απεικόνιση του κουτιού και ένα `onTap`

callback για τη διαχείριση της λειτουργικότητας κατά το πάτημα. Η υλοποίηση αυτή, αν και λειτουργική, ήταν εξαιρετικά υπολογιστικά βαριά, ειδικά για μεγαλύτερα παζλ, όπου ο αριθμός των κελιών και κατ' επέκταση ο αριθμός των widgets αυξανόταν εκθετικά. Αυτό οδήγησε σε καθυστερήσεις τόσο κατά την κύλιση όσο και κατά τη μετάβαση από τη σελίδα επίλυσης στη σελίδα δημιουργίας ή λίστας παζλ.

Για την επίλυση του προβλήματος, το `GridView` αφαιρέθηκε εξ ολοκλήρου και αντικαταστάθηκε από μια πιο αποδοτική προσέγγιση. Αντί για ξεχωριστά widgets για κάθε κελί, δημιουργήθηκε ένας *CustomPainter* για ολόκληρο το πλέγμα. Ο νέος *CustomPainter* ανέλαβε τη σχεδίαση όλων των κελιών του πλέγματος ταυτόχρονα, χρησιμοποιώντας τον ήδη υπάρχοντα painter που είχε δημιουργηθεί για τα ξεχωριστά κουτιά, με μικρές τροποποιήσεις. Επιπλέον, αναπτύχθηκε μια νέα λειτουργικότητα `onTap` και `onDrag`, η οποία εντόπιζε ποιο κελί είχε πατηθεί ή σύρθηκε μέσα στο πλέγμα και επέτρεπε την άμεση επεξεργασία αυτού του συγκεκριμένου κελιού.

Η αλλαγή αυτή είχε σημαντικό αντίκτυπο στην απόδοση. Η χρήση ενός ενιαίου painter μείωσε δραστικά τον αριθμό των widgets που έπρεπε να διαχειριστεί η εφαρμογή, ελαφρύνοντας έτσι το UI. Η κύλιση στη σελίδα έγινε απολύτως ομαλή, ενώ η μετάβαση από τη μία σελίδα στην άλλη ήταν άμεση, χωρίς καθυστερήσεις ή κολλήματα. Επιπλέον, η συνολική εμπειρία του χρήστη βελτιώθηκε, καθώς η εφαρμογή ανταποκρινόταν πλέον χωρίς διακοπές.

Αξίζει να σημειωθεί ότι η αλλαγή αυτή είχε και έναν θετικό έμμεσο αντίκτυπο στον χρόνο επίλυσης του παζλ. Με τη μείωση των υπολογιστικών απαιτήσεων και τη χρήση μικρότερης μνήμης RAM από τις λειτουργίες σχεδίασης, απελευθερώθηκαν περισσότεροι πόροι για τη λειτουργία του αλγορίθμου επίλυσης. Αυτό συνέβαλε στη βελτίωση του χρόνου εκτέλεσης, καθώς το σύστημα μπορούσε να αξιοποιήσει τη διαθέσιμη μνήμη πιο αποδοτικά.

Η αντικατάσταση του `GridView` με έναν ενιαίο *CustomPainter* και η προσθήκη μιας προσαρμοσμένης λειτουργικότητας για την ανίχνευση των ενεργειών του χρήστη στο πλέγμα αποτελούν χαρακτηριστικό παράδειγμα του πώς οι βελτιώσεις στο UI μπορούν να επηρεάσουν θετικά τόσο την εμπειρία του χρήστη όσο και την απόδοση του αλγορίθμου. Με την αλλαγή αυτή, η εφαρμογή έγινε πιο ομαλή, πιο αποδοτική και πιο ευχάριστη στη χρήση.

12.2.2.11 Αναδιοργάνωση και Τεκμηρίωση Κώδικα

Στο τέλος της ανάπτυξης της εφαρμογής, πραγματοποιήθηκε μια εκτεταμένη διαδικασία αναδιοργάνωσης και τεκμηρίωσης του κώδικα. Ο κώδικας της εφαρμογής διαχωρίστηκε ακόμη περισσότερο σε επιμέρους συναρτήσεις, οργανώθηκαν τα αρχεία σε πιο λογικές δομές, ενώ οι συναρτήσεις και τα αντικείμενα μετονομάστηκαν ώστε να αποτυπώνουν καλύτερα τον σκοπό και τη λειτουργία τους. Παράλληλα, διαγράφηκε οποιοδήποτε περιττό ή αχρησιμοποίητο τμήμα του κώδικα, εξαλείφοντας τα υπολείμματα προηγούμενων υλοποιήσεων.

Ένα από τα σημαντικότερα μέρη αυτής της διαδικασίας ήταν η προσθήκη τεκμηρίωσης και σχολίων σε κάθε αρχείο του έργου. Κάθε λειτουργία, κάθε τάξη και κάθε κρίσιμο σημείο του κώδικα συνοδεύτηκε από σαφή περιγραφή, τόσο στη γενική τεκμηρίωση όσο και με σχόλια ανάμεσα των γραμμών (*inline comments*). Δεν υπήρξε κανένα αρχείο που να μην περάσει από αυτό το στάδιο. Αυτή η προσπάθεια δεν ήταν απλώς μια τυπική διαδικασία — ήταν μια βαθιά ανασκόπηση του κώδικα. Η ανάγκη να περιγραφεί τι κάνει κάθε τμήμα του προγράμματος με ακρίβεια με ανάγκασε να επανεξετάσω και να κατανοήσω εκ νέου το έργο σε βάθος.

Κατά τη διάρκεια αυτής της διαδικασίας, αποκαλύφθηκαν λεπτομέρειες που είχαν παραμεληθεί ή μικρά σημεία που μπορούσαν να βελτιωθούν περαιτέρω. Οι αναθεωρήσεις αυτές, παρότι λεπτές, ενίσχυσαν ακόμα περισσότερο τη συνολική ποιότητα του έργου. Επιπλέον, η οργανωμένη τεκμηρίωση κατέστησε τον κώδικα προσβάσιμο και κατανοητό όχι μόνο σε μελλοντικούς συνεργάτες ή προγραμματιστές, αλλά και σε εμένα την ίδια. Η κατανόηση των σκέψεών μου και της λειτουργικότητας κάθε τμήματος έγινε πιο άμεση, εξοικονομώντας χρόνο από μελλοντικές επεξεργασίες ή διορθώσεις.

Αυτή η διαδικασία, παρότι χρονοβόρα — χρειάστηκαν αρκετές ημέρες, ακόμη και εβδομάδες, για να ολοκλη-

ρωθεί — αποδείχθηκε εξαιρετικά πολύτιμη. Είμαι απόλυτα ικανοποιημένη με την απόφαση να αφιερώσω αυτόν τον χρόνο στην τεκμηρίωση και αναδιοργάνωση, καθώς αποτέλεσε το τελευταίο αλλά ζωτικής σημασίας βήμα για να μετατραπεί η εφαρμογή σε ένα ολοκληρωμένο, επαγγελματικό έργο, έτοιμο να υποστηρίξει οποιαδήποτε μελλοντική εξέλιξη ή συνεργασία.

12.2.2.12 Επίλογος

Η πορεία ανάπτυξης και βελτιστοποίησης του αλγορίθμου επίλυσης Εικονόσταυρων ήταν μια διαδρομή γεμάτη ανακαλύψεις, προκλήσεις και εξέλιξη. Κάθε βήμα, από τις πρώτες προσπάθειες μείωσης των χρόνων εκτέλεσης έως την ολοκλήρωση της τεκμηρίωσης και της αναδιοργάνωσης του κώδικα, συνέβαλε στη μετατροπή ενός λειτουργικού εργαλείου σε μια πραγματικά αποδοτική και επαγγελματική εφαρμογή.

Από τις πρώτες τροποποιήσεις, όπως η αφαίρεση των εκτυπώσεων (`print`) και η εισαγωγή *caching*, μέχρι τις πιο προηγμένες τεχνικές, όπως η χρήση *Isolates*, οι δυναμικοί έλεγχοι θέσεων και η αντικατάσταση του `GridView` με έναν ενιαίο *CustomPainter*, κάθε βήμα είχε στόχο να αντιμετωπίσει συγκεκριμένα προβλήματα και να εκμεταλλευτεί κάθε ευκαιρία για βελτίωση. Οι στρατηγικές ταξινόμησης των δεδομένων, η ομαδοποίηση των διαδικασιών και η αξιοποίηση παράλληλων διεργασιών ανέδειξαν τη σημασία της συνδυαστικής προσέγγισης στην ανάπτυξη αλγορίθμων.

Τα αποτελέσματα αυτής της διαδικασίας ήταν πολυδιάστατα. Ο χρόνος επίλυσης μειώθηκε δραματικά, με ορισμένα παζλ να ολοκληρώνονται σε δευτερόλεπτα, ενώ η εμπειρία του χρήστη αναβαθμίστηκε, προσφέροντας μια πιο ομαλή, αποκριτική και ευχάριστη διεπαφή. Η εφαρμογή, πλέον, όχι μόνο επιλύει με ακρίβεια και ταχύτητα ακόμα και τα πιο σύνθετα παζλ, αλλά το κάνει με τρόπο που ενθουσιάζει και ικανοποιεί τον χρήστη.

Η ολοκλήρωση του έργου με την αναδιοργάνωση και τεκμηρίωση του κώδικα έθεσε τη βάση για μελλοντική εξέλιξη, υποστήριξη και συνεργασία. Αυτή η διαδικασία όχι μόνο εξασφάλισε τη διατήρηση της γνώσης και της κατανόησης του έργου, αλλά παρείχε και την ευκαιρία για την ανακάλυψη μικρών σημείων βελτίωσης που μπορεί να είχαν παραβλεφθεί.

Η εφαρμογή δεν είναι απλώς μια επίδειξη τεχνικής ικανότητας, αλλά ένα παράδειγμα του πώς η συστηματική εργασία, η προσοχή στη λεπτομέρεια και η αφοσίωση στην ποιότητα μπορούν να οδηγήσουν σε εξαιρετικά αποτελέσματα. Αυτή η εμπειρία δεν αποτελεί απλώς ένα ολοκληρωμένο έργο αλλά και ένα μάθημα για το πώς η συνεχής αναθεώρηση και η εστίαση στη βελτίωση είναι θεμελιώδεις για την επιτυχία κάθε δημιουργικού εγχειρήματος.

12.3 Ανάλυση επιδόσεων

12.3.1 Εισαγωγή

Η ενότητα αυτή επικεντρώνεται στη μελέτη των χρόνων εκτέλεσης, του αριθμού των ελέγχων και των λοιπών κρίσιμων παραμέτρων που καθορίζουν την αποτελεσματικότητα της εφαρμογής. Στόχος είναι να αναδειχθεί πώς οι βελτιωτικές επεμβάσεις (π.χ. προσωρινή αποθήκευση, τροποποιήσεις στον τρόπο ελέγχου των θέσεων, παράλληλη εκτέλεση) επηρεάζουν τη συνολική απόδοση του συστήματος, όπως αυτή αποτυπώνεται σε ενδεικτικά σενάρια επίλυσης.

Για τον σκοπό αυτό, παρουσιάζονται αναλυτικά δεδομένα από διαφορετικούς πίνακες, όπου συγκεντρώνονται στοιχεία όπως ο χρόνος επίλυσης, τα βήματα που απαιτήθηκαν, ο αριθμός των γραμμών και κελιών που ελέγχθηκαν, οι αναδρομικές επαναλήψεις και οι πληροφορίες προσωρινής αποθήκευσης. Καθένα από τα σενάρια που εξετάζονται (όπως τα «Swing», «Probably Not», «Cat» και «Dancer») φωτίζει ξεχωριστές πτυχές του προβλήματος και καταδεικνύει πώς η σταδιακή ενσωμάτωση τεχνικών βελτιστοποίησης μπορεί να μειώσει αισθητά το υπολογιστικό

κόστος της επίλυσης ενός εικονόσταυρου.

Η ανάλυση αυτή, πέρα από την παράθεση αριθμητικών μεγεθών, λειτουργεί και ως ένα μέσο ποιοτικής αξιολόγησης. Μέσω της παρατήρησης των μεταβολών στα αποτελέσματα, αποτυπώνεται η συμβολή κάθε βελτίωσης στην ταχύτητα και την αξιοπιστία της εφαρμογής. Έτσι, επιβεβαιώνεται ότι η σταδιακή, δομημένη ανάπτυξη του αλγορίθμου οδήγησε σε μια αποδοτική και εύχρηστη λύση, η οποία μπορεί να ανταποκριθεί με επιτυχία σε πληθώρα σεναρίων και μεγεθών εικονόσταυρων.

12.3.2 Δεδομένα πινάκων

Στους παρακάτω πίνακες καταγράφονται τα δεδομένα που συλλέχθηκαν κατά τη διάρκεια της ανάπτυξης και των βελτιστοποιήσεων. Τα αποτελέσματα παρουσιάζονται αναλυτικά για τέσσερα διαφορετικά σενάρια, επισημαίνοντας τις αλλαγές που πραγματοποιήθηκαν σε κάθε στάδιο.

12.3.2.1 Καταγεγραμμένα Δεδομένα

Κάθε πίνακας περιλαμβάνει:

- **Αριθμό Βελτίωσης**: Για να γίνει η αντιστοίχιση με τη βελτίωση που προστέθηκε τελευταία.
- **Χρόνος Εκτέλεσης**: Η συνολική διάρκεια ολοκλήρωσης του αλγορίθμου.
- **Συνολικά Βήματα**: Ο αριθμός των βημάτων που απαιτήθηκαν.
- **Ελεγμένες Γραμμές**: Πλήθος γραμμών που εξετάστηκαν.
- **Ελεγμένα Κουτιά**: Πλήθος κουτιών γραμμής που εξετάστηκαν.
- **Αναδρομικές Επαναλήψεις**: Ο αριθμός των επαναλήψεων κατά την εκτέλεση.
- **Δεδομένα Προσωρινής Αποθήκευσης**: Πλήθος δεδομένων που αποθηκεύτηκαν προσωρινά για τη μείωση του φόρτου.

12.3.2.2 Βελτιώσεις

Οι αριθμοί που αναφέρονται παρακάτω αντιστοιχούν στα δεδομένα που περιγράφονται στους πίνακες που ακολουθούν:

1. **Πρώτη εμφάνιση χρονομέτρου**: Τα δεδομένα της πρώτης εμφάνισης χρονομέτρου αντιπροσωπεύουν την πρώτη καταγραφή δεδομένων του αρχικού παζλ που εκτυπώθηκε. (Αναφορά: Κεφάλαιο “2.2.1 Πρόοδος προς την Πρώτη Ολοκληρωμένη Λύση”).
2. **Αφαίρεση εκτυπώσεων κονσόλας (`print`)**: (Αναφορά: Κεφάλαιο “2.2.2.2”).
3. **Εκκίνηση και τερματισμό ελέγχου σε “?”**: (Αναφορά: Κεφάλαιο “2.2.2.3”).
4. **Ομαδοποίηση συμπλήρωσης κουτιών**: (Αναφορά: Κεφάλαιο “2.2.2.5”).
5. **Προσθήκη προσωρινής αποθήκευσης (*caching*)**: (Αναφορά: Κεφάλαιο “2.2.2.6”).
6. **Προσθήκη `minStartingPoint`**: (Αναφορά: Κεφάλαιο “2.2.2.7”).
7. **Προσθήκη `maxEndingPoint`**: (Αναφορά: Κεφάλαιο “2.2.2.7”).
8. **Χρήση *Isolate* και *Workers* και μικρών διορθώσεων**: (Αναφορά: Κεφάλαιο “2.2.2.8”).
9. **Αύξουσα ταξινόμηση**: (Αναφορά: Κεφάλαιο “2.2.2.9”).
10. **Φθίνουσα ταξινόμηση**: (Αναφορά: Κεφάλαιο “2.2.2.9”).

12.3.2.3 Ορισμοί Στηλών Πινάκων

Για την καλύτερη κατανόηση των πινάκων που ακολουθούν, παρακάτω παρουσιάζονται οι σύντομες ονομασίες των στηλών μαζί με τις πλήρεις τους περιγραφές:

Πίνακας 12.1: Ορισμοί Στηλών

Σύντομο Όνομα	**Πλήρες Όνομα**
Βελτ.	Βελτίωση
Χρόνος	Χρόνος Εκτέλεσης
Βήματα	Συνολικά Βήματα
Γραμμές	Ελεγμένες Γραμμές
Κουτιά	Ελεγμένα Κουτιά
Αναδρομές	Αναδρομικές Επαναλήψεις
ΔΠΑ	Δεδομένα Προσωρινής Αποθήκευσης

12.4 Πίνακες επιδόσεων

12.4.1 Εισαγωγή

Στην παρούσα ενότητα εξετάζονται τα δεδομένα που παρουσιάζονται στους πίνακες της Κεφάλαιο 12.4, με σκοπό να αποτυπωθούν οι συσχετίσεις και τα συμπεράσματα που προκύπτουν από την ανάλυσή τους. Η οργάνωση των πληροφοριών σε πίνακες επιτρέπει την *σαφή* επισκόπηση των μετρήσεων και τη διευκόλυνση συγκριτικών προσεγγίσεων, συμβάλλοντας παράλληλα στη διερεύνηση τυχόν αποκλίσεων ή επαναλαμβανόμενων μοτίβων.

Η έμφαση δίνεται στη μεθοδολογία που ακολουθήθηκε για τη συλλογή και επεξεργασία των δεδομένων, καθώς και στην ακρίβεια των μετρήσεων που παρατίθενται στους πίνακες. Κάθε καταχώριση αντιπροσωπεύει ένα σημείο αναφοράς, το οποίο λειτουργεί ως βάση για περαιτέρω **ποιοτική** και **ποσοτική** ανάλυση. Με αυτόν τον τρόπο, διευκολύνεται η ανάδειξη των κρίσιμων παραγόντων που επηρεάζουν την ερευνητική διαδικασία.

Παράλληλα, παρουσιάζονται ενδεικτικά παραδείγματα που φωτίζουν τις τάσεις που αναπτύσσονται μέσα από τα δεδομένα, επιτρέποντας στον αναγνώστη να διακρίνει τα σημεία που χρήζουν ιδιαίτερης προσοχής. Η οπτικοποίηση των αποτελεσμάτων σε πίνακες συμβάλλει στη διερεύνηση πιθανών *αλληλεπιδράσεων*, αλλά και στη διαμόρφωση υποθέσεων οι οποίες μπορούν να εξεταστούν περαιτέρω σε μεταγενέστερα στάδια της έρευνας.

Τέλος, δίνεται έμφαση στις επιπτώσεις που έχουν τα συγκεκριμένα ευρήματα στην ερμηνεία και την αξιολόγηση του συνόλου της μελέτης. Η κατανόηση της μεθοδολογικής προσέγγισης και η προσεκτική ανάγνωση των πινάκων διασφαλίζουν μια **συνολική εικόνα** των αποτελεσμάτων, καθιστώντας εφικτή την εξαγωγή τεκμηριωμένων συμπερασμάτων και ανοίγοντας τον δρόμο για περαιτέρω βελτιστοποίηση των ερευνητικών μεθόδων.

12.4.2 Εικονόσταυρο “Swing”

Το σενάριο ”Swing” απαιτούσε μεγαλύτερη υπολογιστική δύναμη λόγω της καταγραφής δεδομένων. Στα αρχικά στάδια, οι χρόνοι εκτέλεσης ήταν υψηλοί, όμως η σταδιακή εφαρμογή βελτιστοποιήσεων μείωσε δραματικά τις απαιτήσεις. Η προσθήκη προσωρινής αποθήκευσης (*caching*) και η ομαδοποίηση ελέγχων έπαιξαν κρίσιμο ρόλο στη βελτίωση της απόδοσης.

Πίνακας 12.2: Ανάλυση επιδόσεων για το σενάριο ”Swing”

Βελτ.	Χρόνος	Βήματα	Γραμμές	Κουτιά	Αναδρομές	ΔΠΑ
1	-	-	-	-	-	-
2	0:09:09.007	-	-	-	-	-
3	0:08:26.173	-	-	-	-	-
4*	0:20:40.070	-	1.952	240.615	219.502.834	-
5**	0:10:26.453	-	1.952	240.615	220.739.398	335.243
6***	0:06:18.932	-	1.952	178.985	136.327.896	269.056
7****	0:02:32.803	-	1.952	93.725	40.602.832	161.221
8	0:00:18.577	861	2.150	93.725	40.602.832	161.221
9	0:00:02.435	868	2.115	99.184	1.604.926	160.779
10	0:00:02.162	856	2.115	91.057	1.358.236	146.137

Σημειώσεις:

- *: Ο χρόνος εκτέλεσης χωρίς την καταγραφή των δεδομένων ήταν 0:08:29.728
- **: Ο χρόνος εκτέλεσης χωρίς την καταγραφή των δεδομένων ήταν 0:07:21.925
- ***: Ο χρόνος εκτέλεσης χωρίς την καταγραφή των δεδομένων ήταν 0:04:25.311
- ****: Ο χρόνος εκτέλεσης χωρίς την καταγραφή των δεδομένων ήταν 0:01:59.482

12.4.3 Εικονόσταυρο ”Probably Not”

Το σενάριο ”Probably Not” χαρακτηρίζεται από αυξημένη αναδρομική επεξεργασία. Στα αρχικά στάδια, οι χρόνοι εκτέλεσης και ο αριθμός των ελέγχων ήταν ιδιαίτερα υψηλοί. Ωστόσο, οι εφαρμογές βελτιώσεων όπως η αναδιοργάνωση ελέγχων και η εστίαση σε κρίσιμα σημεία του αλγορίθμου συνέβαλαν στη δραματική μείωση τόσο του χρόνου όσο και του φόρτου επεξεργασίας.

Πίνακας 12.3: Ανάλυση επιδόσεων για το σενάριο ”Probably Not”

Βελτ.	Χρόνος	Βήματα	Γραμμές	Κουτιά	Αναδρομές	ΔΠΑ
1	-	-	-	-	-	-
2	0:00:56.981	-	-	-	-	-
3	0:00:53.389	-	-	-	-	-
4	0:01:18.495	-	846	142.528	25.001.096	-
5	0:00:17.418	-	846	16.456	2.215.880	24.378
6	0:00:13.073	-	846	13.837	2.233.602	21.585
7	0:00:11.171	-	846	11.218	1.529.850	18.356
8	0:00:01.373	217	1.246	10.644	1.351.644	17.592
9	0:00:00.979	219	1.224	10.119	1.103.130	16.308
10	0:00:00.853	217	1.224	9.178	1.057.440	14.927

12.4.4 Εικονόσταυρο “Cat”

Το σενάριο ”Cat” περιλάμβανε μικρότερη υπολογιστική πολυπλοκότητα, με αποτέλεσμα χαμηλούς χρόνους εκτέλεσης ήδη από τα αρχικά στάδια. Η εφαρμογή τεχνικών βελτιστοποίησης, όπως η προσωρινή αποθήκευση και η

βελτιστοποίηση αναδρομικών ελέγχων, μείωσε περαιτέρω τον απαιτούμενο χρόνο, βελτιώνοντας την απόδοση και τη διαχείριση των δεδομένων.

Πίνακας 12.4: Ανάλυση επιδόσεων για το σενάριο ”Cat”

Βελτ.	**Χρόνος**	**Βήματα**	**Γραμμές**	**Κουτιά**	**Αναδρομές**	**ΔΠΑ**
1	0:00:01.551	-	-	-	-	-
2	0:00:00.016	-	-	-	-	-
3	0:00:00.016	-	-	-	-	-
4	0:00:00.036	-	47	648	1.118	-
5	0:00:00.031	-	47	416	634	-
6	0:00:00.025	-	47	372	666	-
7	0:00:00.021	-	47	331	620	-
8	0:00:00.042	32	88	299	584	370
9	0:00:00.101	32	80	299	584	360
10	0:00:00.104	34	80	301	528	375

12.4.5 Εικονόσταυρο “Dancer”

Το σενάριο ”Dancer” ήταν το ευκολότερο για τον αλγόριθμο, με τους μικρότερους χρόνους εκτέλεσης. Οι τεχνικές βελτιστοποίησης, όπως η χρήση προσωρινής αποθήκευσης και η ελαχιστοποίηση επαναλήψεων, εφαρμόστηκαν με επιτυχία, μειώνοντας περαιτέρω το χρόνο και το φόρτο επεξεργασίας.

Πίνακας 12.5: Ανάλυση επιδόσεων για το σενάριο ”Dancer”

Βελτ.	**Χρόνος**	**Βήματα**	**Γραμμές**	**Κουτιά**	**Αναδρομές**	**ΔΠΑ**
1	0:00:01.675	-	-	-	-	-
2	0:00:00.017	-	-	-	-	-
3	0:00:00.017	-	-	-	-	-
4	-	-	-	-	-	-
5	0:00:00.028	-	47	470	862	-
6	0:00:00.027	-	47	421	794	-
7	0:00:00.024	-	47	356	710	-
8	0:00:00.072	37	73	319	658	414
9	0:00:00.117	37	65	319	658	406
10	0:00:00.119	37	65	328	654	414

12.4.6 Συμπεράσματα

Η ανάλυση επιβεβαιώνει τη σημαντική βελτίωση της απόδοσης μέσω των εφαρμοσμένων βελτιστοποιήσεων. Η προσωρινή αποθήκευση, οι τροποποιήσεις στον *έλεγχο θέσεων* και η παράλληλη εκτέλεση οδήγησαν σε αισθητή μείωση του χρόνου εκτέλεσης, ενώ παράλληλα ενίσχυσαν την αξιοπιστία. Τα δεδομένα υποδεικνύουν ότι η μείωση του φόρτου επεξεργασίας επιτυγχάνεται **αποτελεσματικά**, τεκμηριώνοντας έτσι την επιτυχία των βελτιώσεων. Τέλος, η σταδιακή ενσωμάτωση αυτών των παρεμβάσεων προσφέρει μια ευέλικτη εφαρμογή, ικανή να ανταποκρίνεται σε διαφορετικά σενάρια επίλυσης εικονόσταυρων.

12.5 Επίδειξη της λειτουργικότητας της εφαρμογής

12.5.1 Εισαγωγή

Σε αυτή την ενότητα παρουσιάζεται η ροή των ενεργειών του χρήστη στην εφαρμογή. Θα περιγραφεί η διαδικασία προβολής της λύσης ενός έτοιμου εικονόσταυρου, καθώς και η δημιουργία και επίλυση ενός νέου εικονόσταυρου από τον χρήστη. Αναλύονται δύο βασικές ενέργειες: η άμεση εμφάνιση λύσης ενός προεγκατεστημένου εικονόσταυρου και η δυνατότητα σχεδιασμού και επίλυσης ενός προσαρμοσμένου εικονόσταυρου.

12.5.2 Πρώτη ενέργεια: Προβολή λύσης έτοιμου εικονόσταυρου

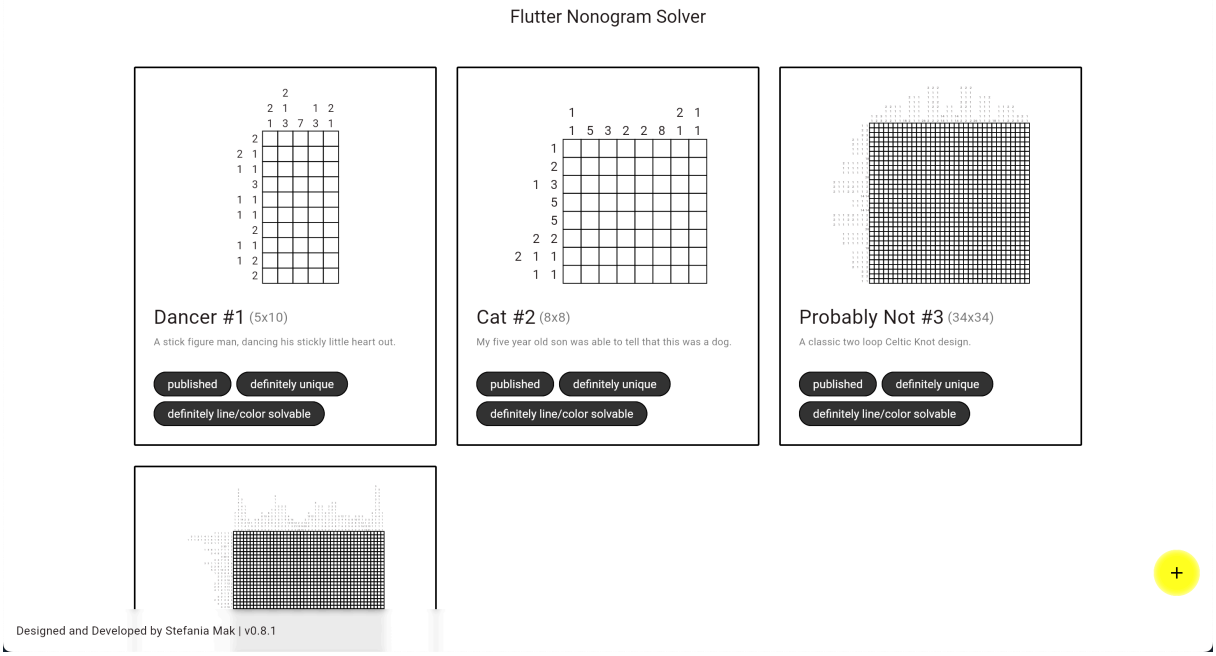
Ο χρήστης μπορεί να επιλέξει έναν έτοιμο εικονόσταυρο και να δει αμέσως τη λύση που παράγεται από τον αλγόριθμο. Μετά την εμφάνιση της λύσης, παρέχεται η δυνατότητα προβολής στατιστικών στοιχείων σχετικά με την εκτέλεση του αλγορίθμου μέσω του τμήματος “Στατιστικά” (`--Stats--`). Ο χρήστης μπορεί επίσης να εμφανίσει παλαιότερες λύσεις χρησιμοποιώντας το συρόμενο ελεγκτή κάτω από το κουμπί “ΕΠΙΛΥΣΗ” (`SOLVE`).

12.5.2.1 Σελίδα Λίστας Έτοιμων Εικοόσταυρων

Η σελίδα λίστα των έτοιμων εικονόσταυρων παρουσιάζει όλα τα διαθέσιμα εικονόσταυρα που έχουν προεγκατασταθεί στην εφαρμογή. Αυτή η σελίδα είναι σχεδιασμένη ώστε να διευκολύνει τον χρήστη στην επιλογή του κατάλληλου εικονόσταυρου για επίλυση, παρέχοντας μια ευδιάκριτη εμφάνιση και λειτουργικά φίλτρα αναζήτησης.

12.5.2.1.1 Αρχική Σελίδα με Διαθέσιμα Έτοιμα Εικονόσταυρα

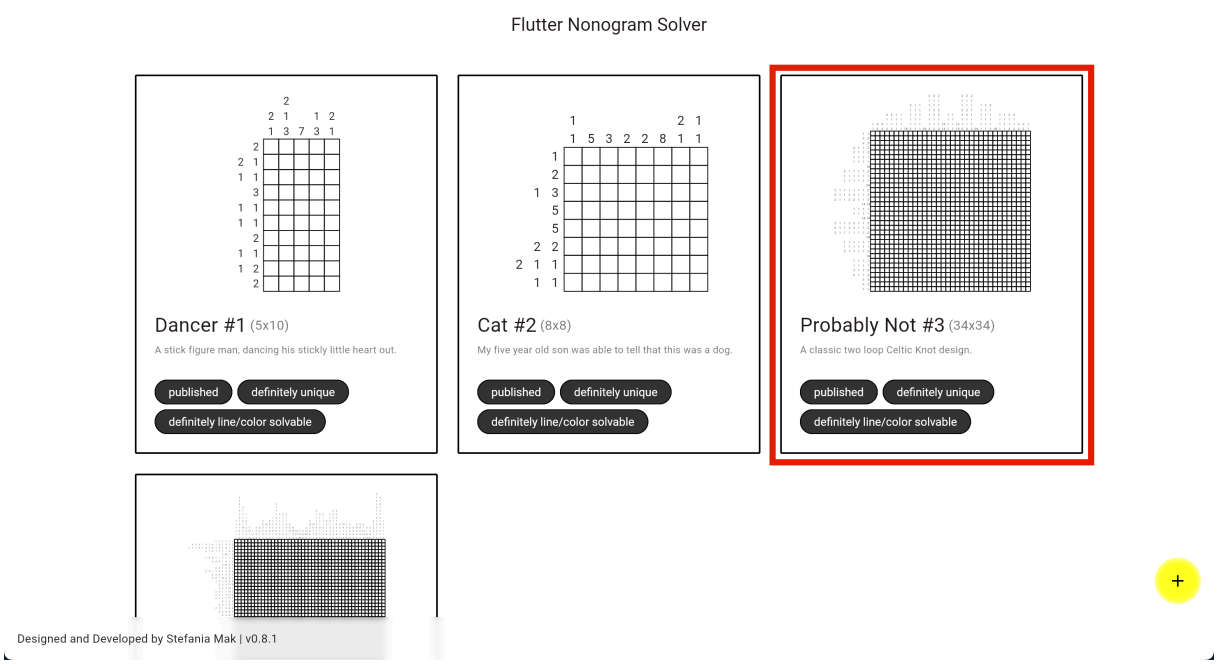
Η αρχική σελίδα εμφανίζει μια οργανωμένη λίστα εικονόσταυρων, με όνομα, δυσκολία και σύντομη προεπισκόπηση πλέγματος, διευκολύνοντας την ταχεία αναζήτηση και επιλογή, ενώ επιτρέπει ταξινόμηση και φιλτράρισμα, σύμφωνα με τις προτιμήσεις του χρήστη.



Σχήμα 12.6: Λίστα έτοιμων εικονόσταυρων με πληροφορίες και προεπισκοπήσεις

12.5.2.1.2 Επιλογή Έτοιμου Εικονόσταυρου από τη Λίστα

Αφού ο χρήστης εντοπίσει τον επιθυμητό εικονόσταυρο στη λίστα, μπορεί να τον επιλέξει με ένα απλό κλικ. Η επιλογή αυτή μεταφέρει τον χρήστη σε μια νέα σελίδα αφιερωμένη στον συγκεκριμένο εικονόσταυρο, όπου θα εμφανιστεί η λύση του αλγορίθμου. Η μετάβαση είναι γρήγορη και ομαλή, χωρίς καθυστερήσεις, διασφαλίζοντας μια ευχάριστη εμπειρία χρήστη. Επιπλέον, η εφαρμογή παρέχει δυνατότητες προεπισκόπησης του εικονόσταυρου πριν την επιλογή του, επιτρέποντας στον χρήστη να δει μια μικρή εικόνα του παζλ και να αποφασίσει αν θέλει να το λύσει ή να επιλέξει κάποιο άλλο.



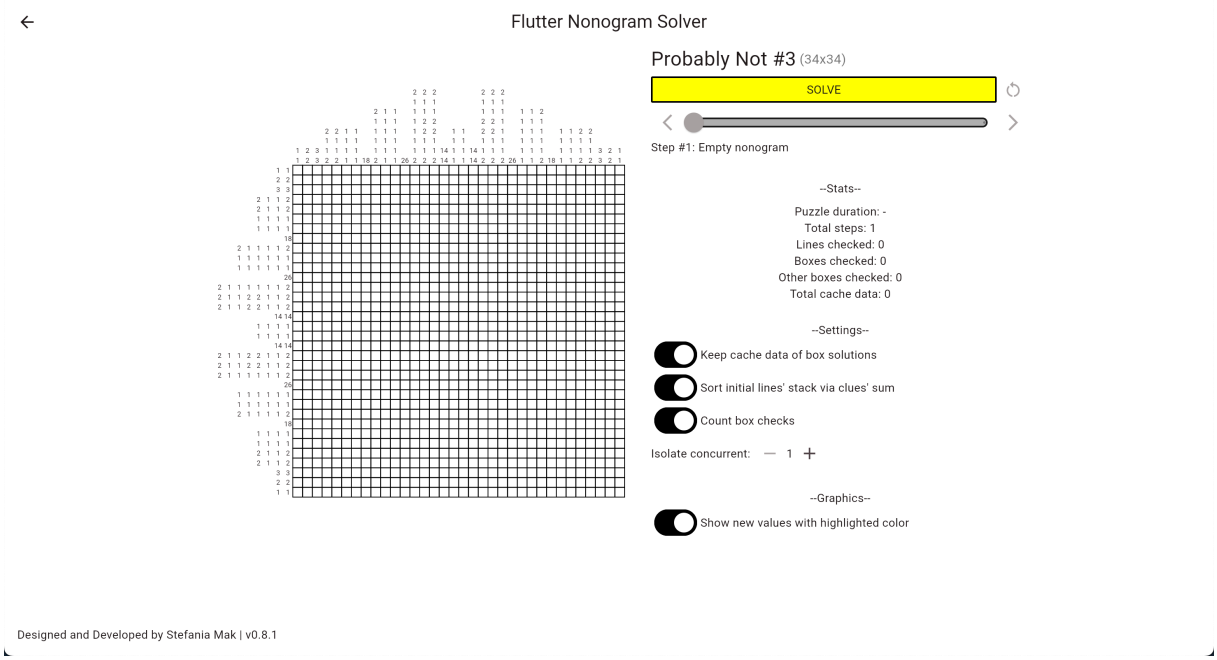
Σχήμα 12.7: Επιλογή ενός έτοιμου εικονόσταυρου από την λίστα

12.5.2.2 Σελίδα Λύσης Εικονόσταυρου

Η σελίδα λύσης προβάλλει τον επιλεγμένο εικονόσταυρο σε μεγαλύτερη μορφή, με βασικές πληροφορίες (τίτλο, αριθμό, μέγεθος) και δύο κουμπιά: «ΕΠΙΛΥΣΗ» που εκτελεί τον αλγόριθμο και διαγραφής που επαναφέρει το πλέγμα. Κατά την επίλυση, εμφανίζεται συρόμενος ελεγκτής προόδου και απενεργοποιούνται οι ρυθμίσεις. Μετά την ολοκλήρωση, ο χρήστης βλέπει στατιστικά (π.χ. αριθμός βημάτων) και μπορεί να επιστρέψει σε προηγούμενα στάδια ή να επαναφέρει το πλέγμα για νέα εκτέλεση χωρίς επανεκκίνηση.

12.5.2.2.1 Σελίδα Επίλυσης Έτοιμου Εικονόσταυρου

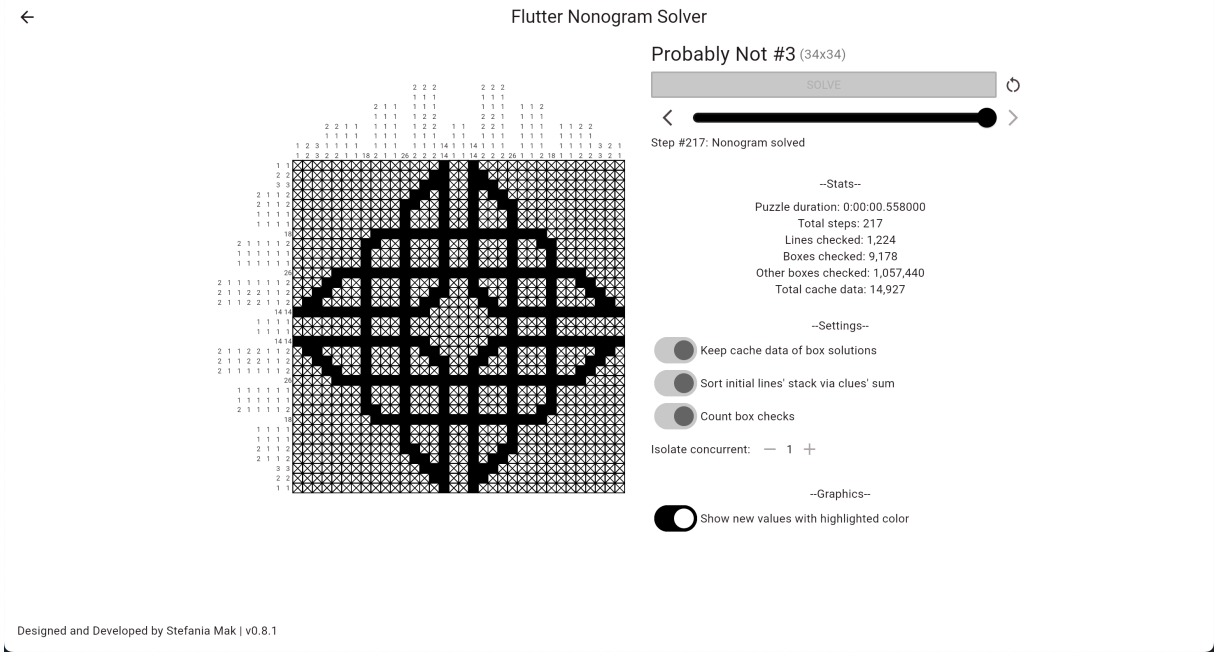
Στη σελίδα λύσης εμφανίζεται ο επιλεγμένος εικονόσταυρος με βασικές πληροφορίες όπως τίτλο, αριθμό και μέγεθος. Κάτω από αυτές, υπάρχουν δύο κουμπιά: το “ΕΠΙΛΥΣΗ” (“SOLVE”) για εκτέλεση του αλγορίθμου και ένα απενεργοποιημένο κουμπί διαγραφής που επαναφέρει το πλέγμα στην αρχική του μορφή. Σε αυτό το στάδιο, ο χρήστης μπορεί να επεξεργαστεί τις ρυθμίσεις του αλγορίθμου, όπως η διατήρηση cache, η καταμέτρηση ελέγχων ή η ταξινόμηση της λίστας στοίβας. Κατά την εκτέλεση, εμφανίζεται ένας συρόμενος ελεγκτής που παρακολουθεί την πρόοδο σε πραγματικό χρόνο και απενεργοποιεί τις επιλογές διαμόρφωσης. Με την ολοκλήρωση της επίλυσης, παρέχονται στατιστικά στοιχεία και η δυνατότητα επαναφοράς του πλέγματος για νέα εκτέλεση με διαφορετικές ρυθμίσεις χωρίς επανεκκίνηση της εφαρμογής.



Σχήμα 12.8: Σελίδα επιλεγμένου εικονόσταυρου

12.5.2.2.2 Τελική Λύση του Αλγορίθμου

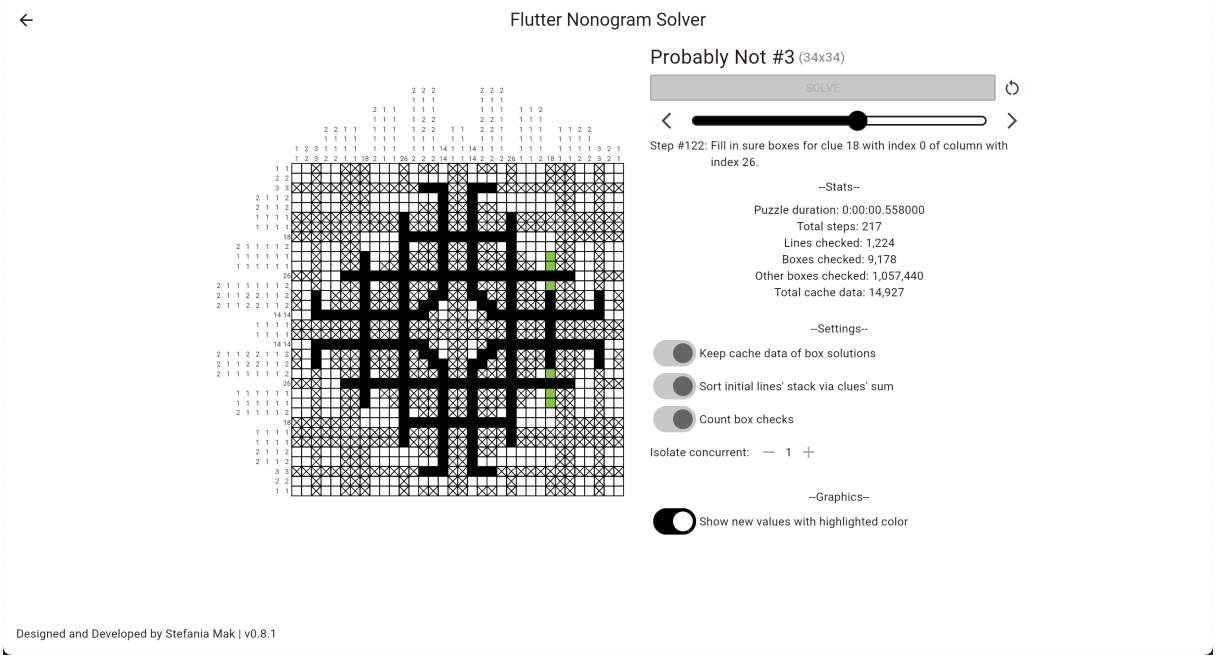
Μετά την ολοκλήρωση της επίλυσης, εμφανίζεται η τελική λύση του εικονόσταυρου. Αυτή η λύση παρουσιάζεται καθαρά και ευανάγνωστα, επιτρέποντας στον χρήστη να επιβεβαιώσει την ορθότητά της. Παράλληλα, εμφανίζονται στατιστικά στοιχεία που περιγράφουν τη διαδικασία επίλυσης, όπως ο συνολικός αριθμός βημάτων και οι αναδρομικές επαναλήψεις.



Σχήμα 12.9: Τελική λύση του αλγορίθμου

12.5.2.2.3 Προβολή Ενδιάμεσου Σταδίου της Λύσης

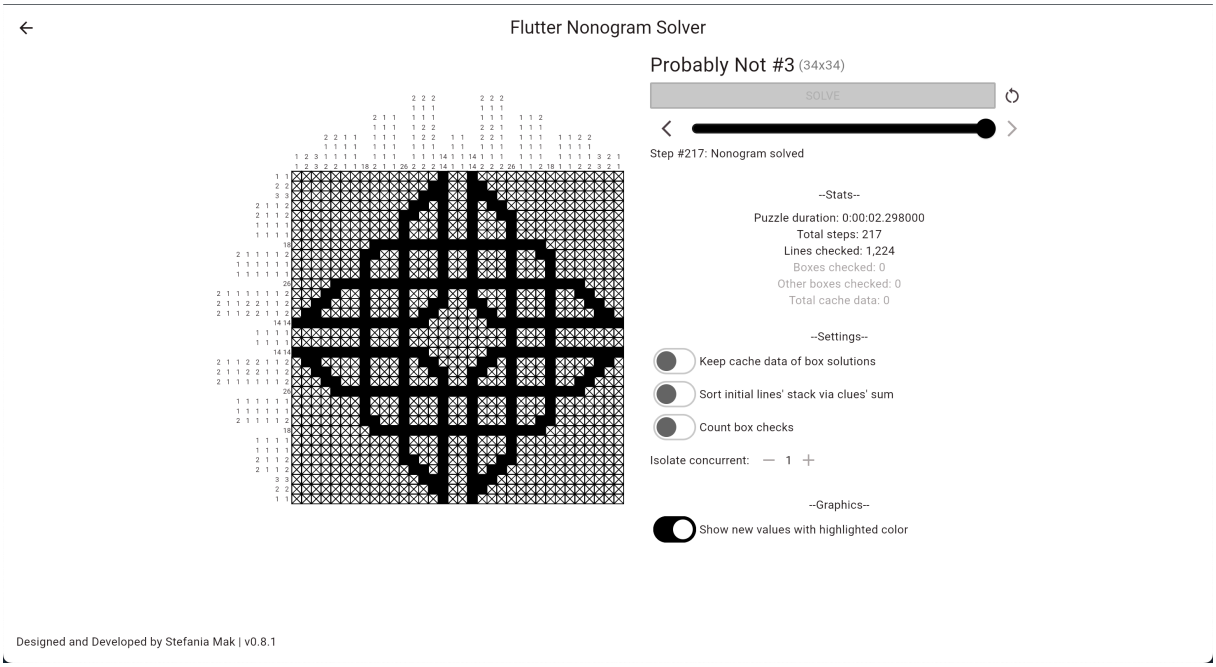
Κατά την επίλυση, ο χρήστης μπορεί να παρακολουθεί βήμα προς βήμα την πορεία, κατανοώντας καλύτερα τη λειτουργία του αλγορίθμου και τις ενέργειες που εκτελούνται.



Σχήμα 12.10: Προβολή ενδιάμεσου σταδίου της λύσης

12.5.2.2.4 Τρέξιμο Λύτη με Απενεργοποιημένες Επιλογές

Τρέχοντας την εφαρμογή με *απενεργοποιημένες* όλες τις επιλογές στις ρυθμίσεις, παρατηρούμε ότι υπάρχει αισθητή διαφοροποίηση στις **τιμές των αποτελεσμάτων**.



Σχήμα 12.11: Τρέξιμο λύτη με απενεργοποιημένες επιλογές

12.5.2.3 Συμπέρασμα Πρώτης Ενέργειας

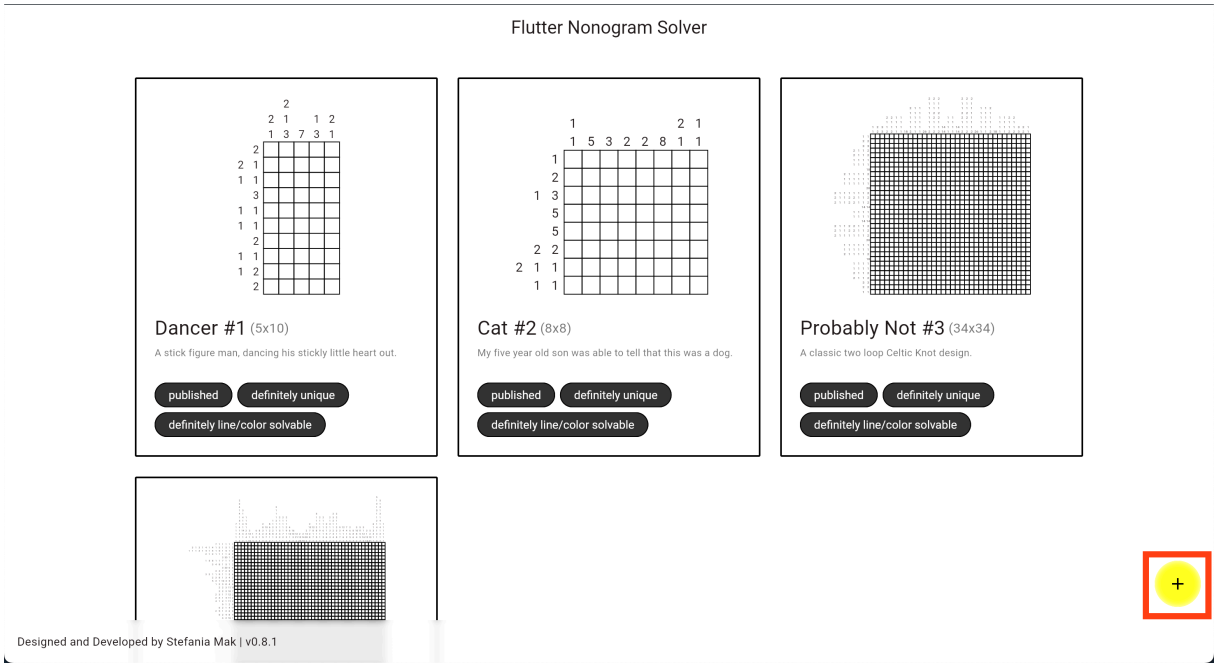
Η διαδικασία προβολής λύσης ενός έτοιμου εικονόσταυρου έχει σχεδιαστεί για να είναι γρήγορη και απλή, παρέχοντας στον χρήστη άμεση πρόσβαση στη λύση του παζλ που επιλέγει. Με την ενσωμάτωση στατιστικών στοιχείων και την δυνατότητα παρακολούθησης της πρόοδου σε πραγματικό χρόνο, η εφαρμογή προσφέρει μια ολοκληρωμένη εμπειρία χρήστη που ενισχύει την κατανόηση και την αξιολόγηση της αποτελεσματικότητας του αλγορίθμου επίλυσης.

12.5.3 Δεύτερη ενέργεια: Σχεδιασμός και αυτόματη επίλυση προσαρμοσμένου εικονόσταυρου

Η δεύτερη ενέργεια της εφαρμογής επικεντρώνεται στη δυνατότητα σχεδιασμού ενός προσαρμοσμένου εικονόσταυρου και στην αυτοματοποιημένη του επίλυση. Αυτή η λειτουργία επιτρέπει στους χρήστες να δημιουργούν μοναδικά παζλ σύμφωνα με τις προτιμήσεις τους, παρέχοντας ταυτόχρονα εργαλεία για την αποτελεσματική επίλυσή τους. Η διαδικασία σχεδιασμού και επίλυσης είναι διαχωρισμένη σε δύο βασικά στάδια: το άνοιγμα της σελίδας δημιουργίας εικονόσταυρου και η δημιουργία του εικονόσταυρου μέσω ζωγραφικής ή απευθείας εισαγωγής στοιχείων.

12.5.3.1 Άνοιγμα σελίδας δημιουργίας εικονόσταυρου

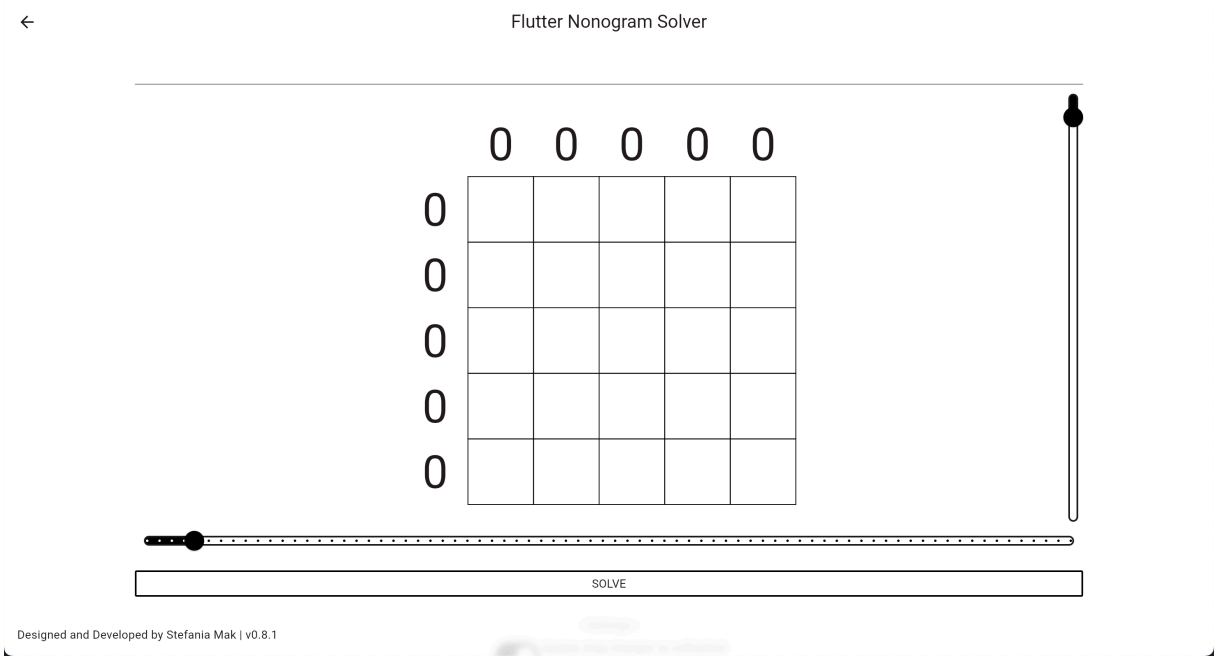
Η διαδικασία ξεκινά από την αρχική σελίδα της εφαρμογής, όπου εμφανίζονται και τα προεγκατεστημένα εικονόσταυρα. Στην κάτω δεξιά γωνία της σελίδας βρίσκεται ένα κίτρινο αιωρούμενο κουμπί δράσης (Floating Action Button) με το σύμβολο «+». Αυτό το κουμπί χρησιμεύει ως σημείο εκκίνησης για τη δημιουργία ενός νέου εικονόσταυρου, επιτρέποντας στον χρήστη να μεταβεί άμεσα στη σελίδα δημιουργίας με ένα απλό πάτημα.



Σχήμα 12.12: Κουμπί εισόδου στη σελίδα δημιουργίας εικονόσταυρου

12.5.3.2 Σελίδα Δημιουργίας Εικονόσταυρου

Πατώντας το κουμπί «+», ο χρήστης μεταφέρεται σε νέα σελίδα δημιουργίας εικονόσταυρου, όπου ορίζει το μέγεθος του πλέγματος και επιλέγει μέθοδο σχεδιασμού, είτε με ζωγραφική στο πλέγμα είτε με εισαγωγή στοιχείων γραμμών και στηλών.



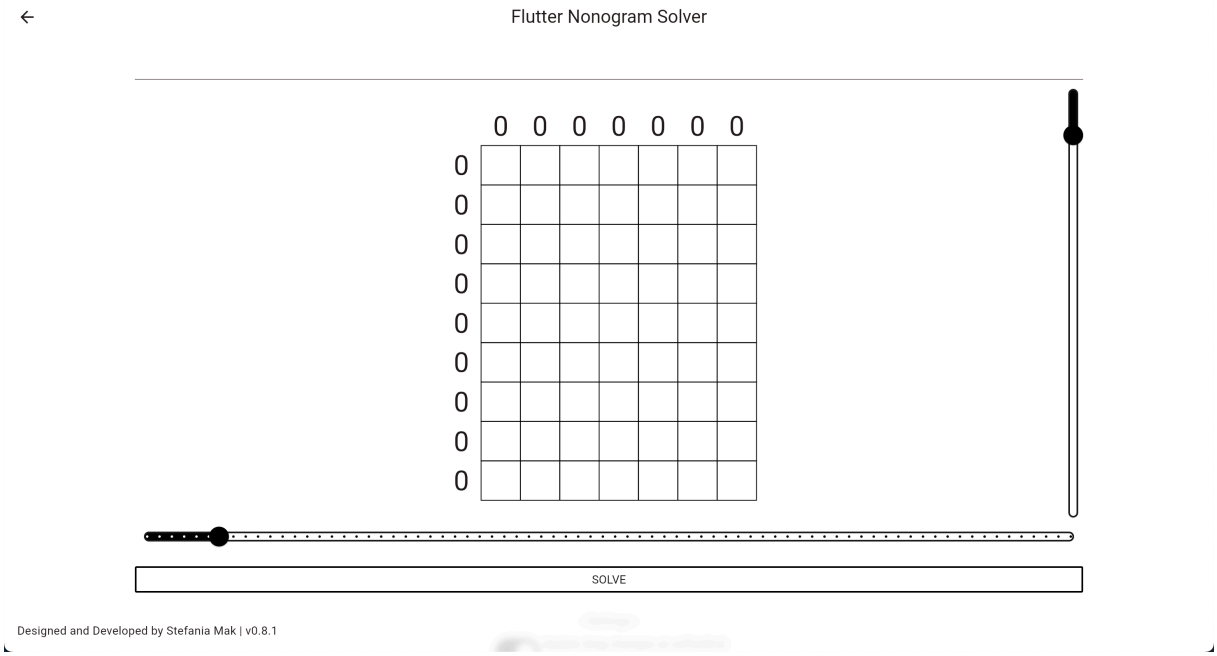
Σχήμα 12.13: Σελίδα δημιουργίας εικονόσταυρου

12.5.3.3 Δημιουργία εικονόσταυρου

Στη σελίδα δημιουργίας, ο χρήστης ρυθμίζει το μέγεθος του πλέγματος και επιλέγει μεταξύ ζωγραφικής ή εισαγωγής γραμμών-στηλών, δημιουργώντας προσαρμοσμένους εικονόσταυρους.

12.5.3.3.1 Προσαρμογή μεγέθους πλέγματος

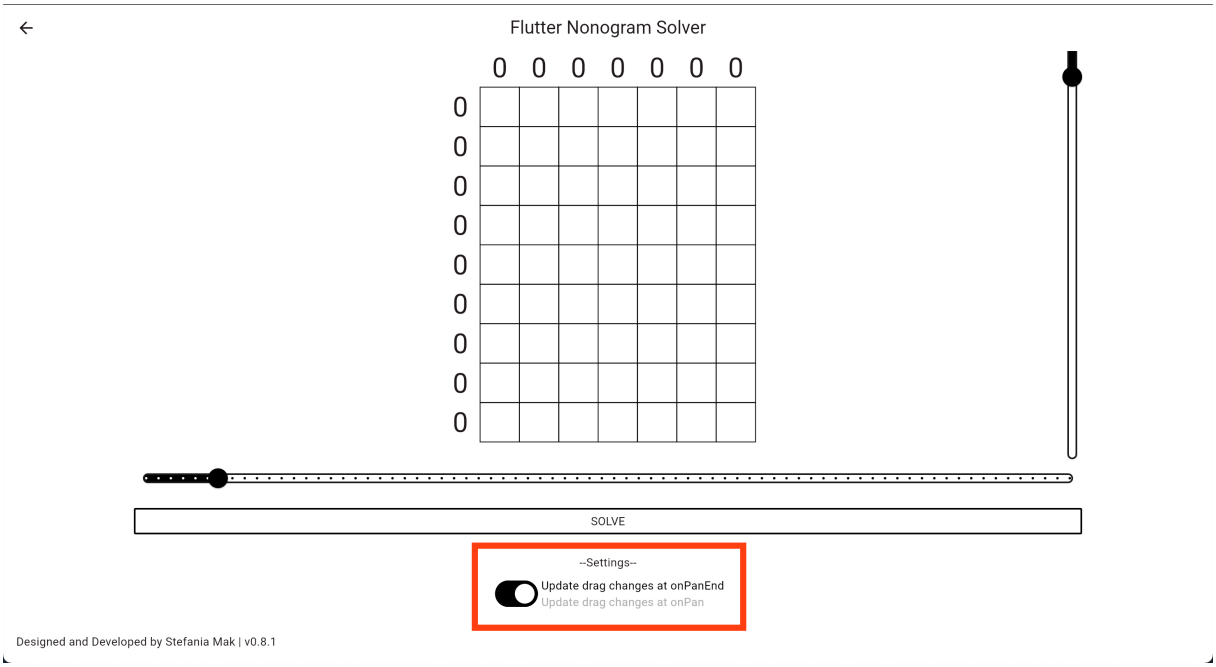
Οι συρόμενοι ελεγκτές δεξιά και κάτω από το πλέγμα επιτρέπουν στον χρήστη να αλλάξει το μέγεθος του εικονόσταυρου οποιαδήποτε στιγμή, διατηρώντας τα δεδομένα και την ακεραιότητα του παζλ.



Σχήμα 12.14: Προσαρμογή μεγέθους πλέγματος από τους συρόμενους ελεγκτές

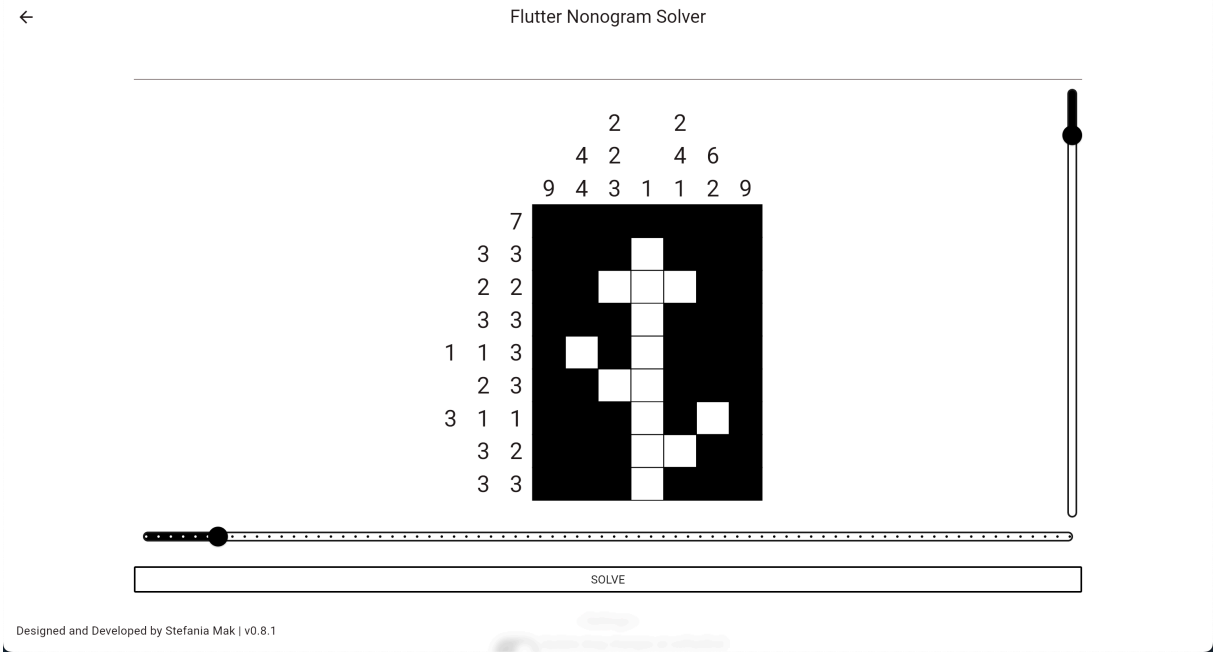
12.5.3.3.2 Δημιουργία εικονόσταυρου με ζωγραφική

Ο χρήστης μπορεί να επιλέξει ανάμεσα σε δύο μεθόδους ανανέωσης, οι οποίες ρυθμίζονται στο κάτω μέρος της σελίδας (Σχήμα 12.15).



Σχήμα 12.15: Ρυθμίσεις ζωγραφικής

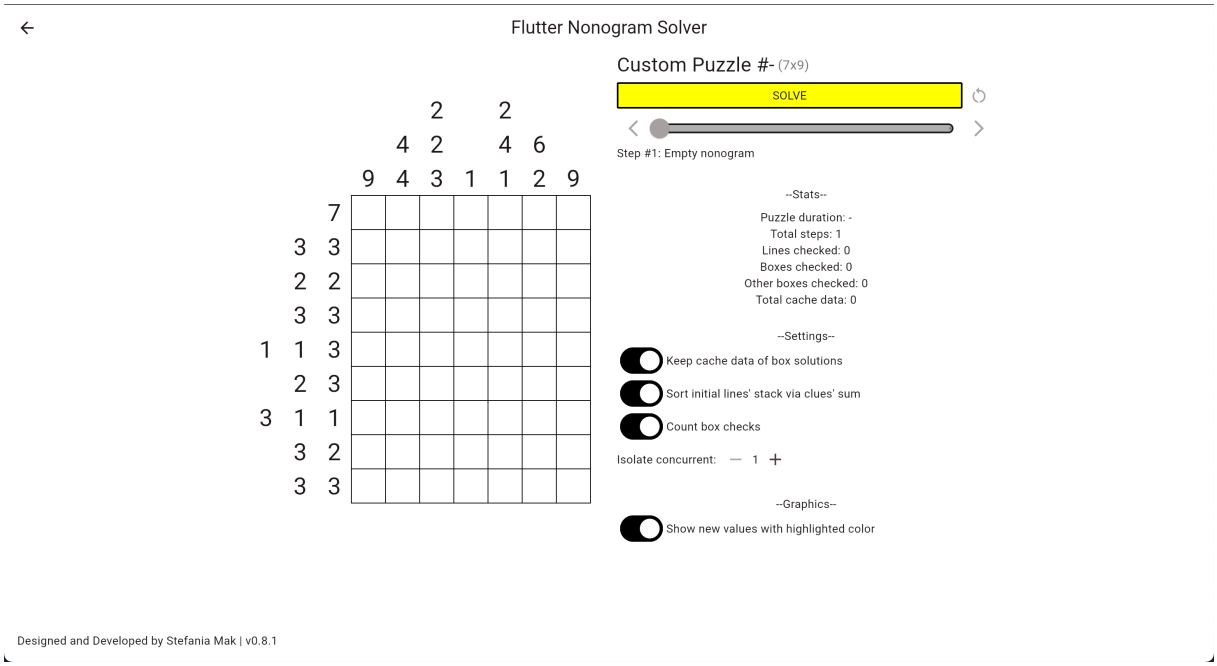
Ο χρήστης σχεδιάζει το μοτίβο με κλικ στο πλέγμα, προσαρμόζοντας παραμέτρους (cache, έλεγχοι, ταξινόμηση) και πατώντας «ΕΠΙΛΥΣΗ» (*SOLVE*) ξεκινά ηαυτόματη επίλυση.



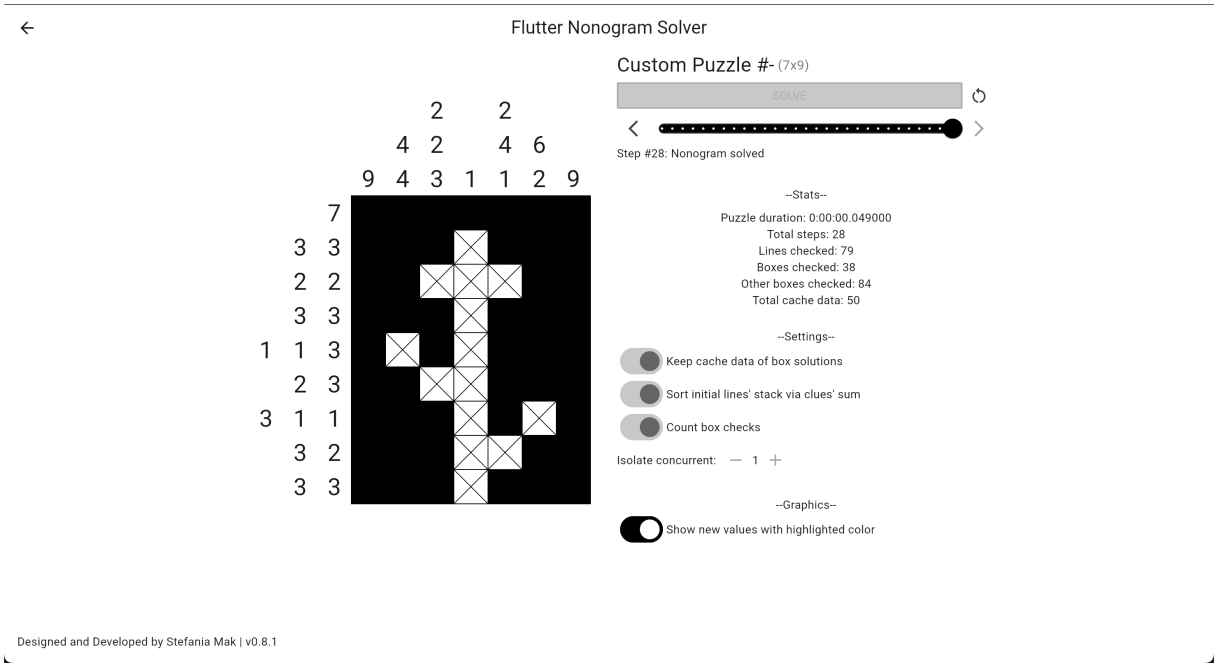
Σχήμα 12.16: Ζωγραφισμένο εικονόσταυρο με το ποντίκι του υπολογιστή

12.5.3.3.3 Λύση Εικονόσταυρου από Ζωγραφική

Με το πάτημα του κουμπιού “ΕΠΙΛΥΣΗ” (“SOLVE”), η εφαρμογή μεταφέρει τον χρήστη στη σελίδα λύσης, όπου εμφανίζεται το σχεδιασμένο πλέγμα σε κενή κατάσταση (Σχήμα 12.17). Σε αυτό το σημείο, η σελίδα λειτουργεί όπως κάθε άλλο εικονόσταυρο με ίδιες επιλογές και ρυθμίσεις. Πατώντας ξανά το κουμπί “ΕΠΙΛΥΣΗ” (“SOLVE”), ο αλγόριθμος συμπληρώνει το εικονόσταυρο, αναπαράγοντας το αρχικά ζωγραφισμένο σχέδιο (Σχήμα 12.18).



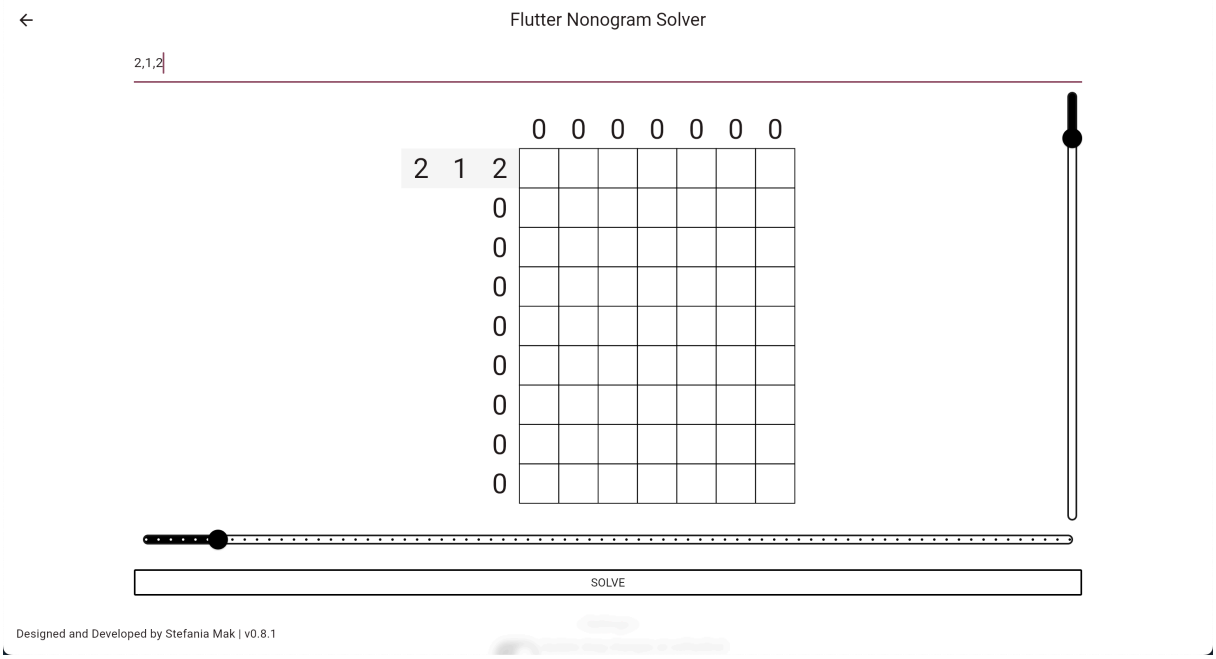
Σχήμα 12.17: Μεταφορά του νέου εικονόσταυρου για αυτόματη επίλυση



Σχήμα 12.18: Λύση του νέου εικονόσταυρου από τον αλγόριθμο

12.5.3.3.4 Δημιουργία Εικονόσταυρου μέσω Στοιχείων

Η εναλλακτική δημιουργία εικονόσταυρου περιλαμβάνει την άμεση εισαγωγή αριθμητικών ενδείξεων για κάθε γραμμή και στήλη, υποδεικνύοντας τη σύνθεση του εικονόσταυρου. Ο χρήστης επιλέγει γραμμή ή στήλη και εισάγει τις ενδείξεις στο πεδίο κειμένου πάνω από το πλέγμα και κάτω από την επικεφαλίδα. Τα νούμερα διαχωρίζονται με κόμματα για να προσδιορίσουν τις ομάδες συμπληρωμένων κουτιών. Μετά την εισαγωγή όλων των γραμμών και στηλών, ο χρήστης πατάει το κουμπί “ΕΠΙΛΥΣΗ” (“SOLVE”) για να ξεκινήσει η αυτόματη επίλυση του εικονόσταυρου βάσει των καταγεγραμμένων ενδείξεων (Σχήμα 12.19).



Σχήμα 12.19: Εναλλακτικός τρόπος δημιουργίας εικονόσταυρου με απευθείας εισαγωγή στοιχείων

12.5.4 Σύνοψη

Συμπερασματικά, η εφαρμογή επιτρέπει στον χρήστη να επιλέγει ένα έτοιμο εικονόσταυρο για άμεση επίλυση ή να δημιουργεί ένα καινούριο, αφήνοντας την επεξεργασία του πλέγματος εντελώς ανοιχτή στις ανάγκες του. Στη συνέχεια, η εφαρμογή αναλαμβάνει την αυτόματη επίλυσή του, προσφέροντας στον χρήστη τη δυνατότητα να παρακολουθήσει τη διαδικασία και να εξετάσει τις διάφορες ρυθμίσεις και ενδιάμεσες καταστάσεις. Με τον τρόπο αυτό, γίνεται ευδιάκριτη τόσο η ευκολία χρήσης της εφαρμογής για γρήγορη προβολή έτοιμων λύσεων, όσο και η ευελιξία που παρέχεται για δημιουργικά και πιο εξειδικευμένα σενάρια.

12.6 Επίλογος

Ολοκληρώνοντας την παρουσίαση των αποτελεσμάτων, διαπιστώνεται ότι η αλληλεπίδραση ανάμεσα στα τεχνικά χαρακτηριστικά του αλγορίθμου και στις ανάγκες του τελικού χρήστη υπήρξε καθοριστική για την επιτυχία του έργου. Οι ενότητες που περιγράφηκαν αναδεικνύουν την αξία της προοδευτικής εξέλιξης: η αρχική μορφή του αλγορίθμου υπέστη διαδοχικές βελτιστοποιήσεις, οι οποίες οδήγησαν σε σημαντική μείωση των χρόνων εκτέλεσης και βελτίωση της γενικότερης αποδοτικότητας. Ταυτόχρονα, η ανάπτυξη μιας ευχάριστης και ευέλικτης διεπαφής χρήστη ενίσχυσε την προσβασιμότητα της εφαρμογής, καλύπτοντας ευρύ φάσμα σεναρίων: από την άμεση επίλυση ενός έτοιμου εικονόσταυρου έως τον δημιουργικό σχεδιασμό ενός καινούριου.

Οι μετρήσεις που καταγράφηκαν επιβεβαιώνουν τόσο την επιστημονική εγκυρότητα όσο και τη βιωσιμότητα του

συστήματος σε πρακτικό επίπεδο. Η ικανότητα του αλγορίθμου να διαχειρίζεται σύνθετα ή μεγάλα παζλ, σε συνδυασμό με την ταχύτατη ανατροφοδότηση και την ευκολία της διεπαφής, καθιστά την εφαρμογή μια πλήρη και αξιόπιστη πρόταση για την επίλυση Εικονόσταυρων. Στο επόμενο κεφάλαιο, θα διερευνηθούν αναλυτικότερα οι συμπερασματικές παρατηρήσεις και οι μελλοντικές προοπτικές που προκύπτουν από αυτήν τη μελέτη, επιβεβαιώνοντας τη δυναμική του έργου για περαιτέρω βελτίωση και επέκταση.

Κεφάλαιο 13ο: Συμπεράσματα και Προτάσεις Βελτίωσης

13.1 Εισαγωγή

Η ενότητα αυτή παρουσιάζει τις παρατηρήσεις και τις προτάσεις βελτίωσης που προέκυψαν από την ανάπτυξη και τη χρήση της εφαρμογής επίλυσης εικονόσταυρων. Η εργασία ανέδειξε σημαντικά δυνατά σημεία, όπως η αποδοτικότητα του αλγορίθμου, η ευκολία χρήσης και η εκπαιδευτική της αξία. Παράλληλα, οι περιορισμοί που εντοπίστηκαν παρέχουν πολύτιμα δεδομένα για τη μελλοντική εξέλιξη και τη διεύρυνση των δυνατοτήτων της εφαρμογής. Με βάση αυτά, η ανάλυση επικεντρώνεται στην αποτίμηση της παρούσας κατάστασης, στις προτεινόμενες βελτιώσεις και στους τρόπους ενίσχυσης της χρησιμότητας της εφαρμογής.

13.2 Ανάλυση Δυνατών Σημείων και Περιορισμών

Η εφαρμογή που αναπτύχθηκε παρουσιάζει μια σειρά από δυνατά σημεία που την καθιστούν καινοτόμο στον τομέα της επίλυσης εικονόσταυρων (Nonograms), καθώς και ορισμένους περιορισμούς που επηρεάζουν τη λειτουργικότητά της.

Ένα από τα μεγαλύτερα πλεονεκτήματα είναι η υλοποίηση ενός αποδοτικού αλγορίθμου επίλυσης, που συνδυάζει τεχνικές λογικής και παράλληλης επεξεργασίας μέσω του μηχανισμού των *isolates*. Αυτό επιτρέπει γρήγορη επεξεργασία δεδομένων, ιδιαίτερα σε παζλ μεσαίου μεγέθους. Οι λύσεις είναι ακριβείς και ολοκληρώνονται χωρίς σφάλματα, ενώ η αρχιτεκτονική του αλγορίθμου εντοπίζει κοινά μοτίβα για αποδοτικότερη χρήση μνήμης και χρόνου.

Η ενσωμάτωση εκπαιδευτικών χαρακτηριστικών είναι επίσης σημαντική. Η προβολή των βημάτων επίλυσης με επεξηγήσεις σε πραγματικό χρόνο επιτρέπει στους χρήστες να κατανοούν τη λογική του αλγορίθμου. Αυτό καθιστά την εφαρμογή όχι μόνο εργαλείο επίλυσης, αλλά και μέσο εκμάθησης των τεχνικών που χρησιμοποιούνται για την επίλυση εικονόσταυρων.

Η επιλογή του Flutter ως πλατφόρμα ανάπτυξης αποτελεί άλλο ένα πλεονέκτημα. Παρέχει τη δυνατότητα ανάπτυξης εφαρμογών που λειτουργούν σε πολλές πλατφόρμες, εξοικονομώντας πόρους και χρόνο. Παρά τα οφέλη, όμως, η χρήση του Flutter επιβάλλει περιορισμούς στη διαχείριση μεγάλων ή σύνθετων παζλ, λόγω αυξημένων απαιτήσεων μνήμης και υπολογιστικής ισχύος.

Παρότι η εφαρμογή προσφέρει δυνατότητα κατανόησης των βημάτων επίλυσης, απουσιάζουν διαδραστικές λειτουργίες σε μεγάλα παζλ, όπως προσαρμογή ενδείξεων σε πραγματικό χρόνο. Επίσης, η διεπαφή χρήστη, αν και λειτουργική, μπορεί να βελτιωθεί με δυναμικά γραφικά και χαρακτηριστικά όπως δημιουργία στατιστικών ή διαμοιρασμός παζλ.

Συνολικά, η εφαρμογή ισορροπεί μεταξύ αποδοτικότητας και ευχρηστίας, αποτελώντας ένα χρήσιμο εργαλείο τόσο για επίλυση όσο και για εκπαίδευση. Παρότι οι περιορισμοί της είναι εμφανείς, οι μελλοντικές βελτιώσεις μπορούν να την καθιερώσουν ως κορυφαία επιλογή στον τομέα της.

13.3 Σημασία των Χαρακτηριστικών της Εφαρμογής

Η σημασία των χαρακτηριστικών της εφαρμογής έγκειται στη μοναδική της ικανότητα να συνδυάζει τη λειτουργικότητα με την **εκπαιδευτική διάσταση**, προσφέροντας μια ολοκληρωμένη εμπειρία στους χρήστες. Ο συνδυασμός εργαλείων επίλυσης και δημιουργίας εικονόσταυρων καθιστά την εφαρμογή ιδανική για ένα ευρύ φάσμα χρηστών, από αρχάριους μέχρι προχωρημένους.

Ένα από τα πιο σημαντικά χαρακτηριστικά της είναι η παροχή **επεξηγήσεων σε κάθε βήμα της λύσης**. Ο χρήστης μπορεί να κατανοήσει τη λογική πίσω από κάθε ενέργεια, καθιστώντας την εφαρμογή ένα χρήσιμο εργαλείο μάθησης. Ο συρόμενος ελεγκτής (**slider**) ενισχύει περαιτέρω την κατανόηση, δίνοντας τη δυνατότητα να ανατρέξει στα βήματα της λύσης με τον δικό του ρυθμό.

Η φιλοξενία στο **Firebase** και η ανάπτυξη με **Flutter** εξασφαλίζουν προσβασιμότητα και λειτουργία σε πολλαπλές πλατφόρμες. Οι χρήστες μπορούν να χρησιμοποιήσουν την εφαρμογή χωρίς πολύπλοκες ρυθμίσεις, κάνοντάς την εύκολα διαθέσιμη και πρακτική. Η δυνατότητα δημιουργίας νέων παζλ προσφέρει μια **εξατομικευμένη εμπειρία**, προάγοντας τη δημιουργικότητα και τη διάδραση μεταξύ χρηστών.

Παρότι υπάρχουν περιορισμοί στο Flutter για βαριές υπολογιστικές λειτουργίες, η εφαρμογή αξιοποιεί τα δυνατά σημεία της πλατφόρμας, προσφέροντας ένα **καλαίσθητο και λειτουργικό γραφικό περιβάλλον**. Οι σχεδιαστικές επιλογές διευκολύνουν τη χρήση και την πλοήγηση από ένα ευρύ κοινό.

Συνολικά, τα χαρακτηριστικά της εφαρμογής την καθιστούν ένα ισχυρό εργαλείο για **επίλυση, εκπαίδευση και δημιουργία**. Με την έμφαση στην προσβασιμότητα και την τεχνολογική καινοτομία, θέτει μια ισχυρή βάση για περαιτέρω ανάπτυξη και βελτιώσεις.

13.4 Πιθανές Βελτιώσεις και Μελλοντικές Κατευθύνσεις

13.4.1 Εισαγωγή

Η εφαρμογή για την επίλυση εικονόσταυρων έχει επιτύχει έναν *υψηλό βαθμό λειτουργικότητας* και *χρηστικότητας*, προσφέροντας μια **ολοκληρωμένη εμπειρία** στους χρήστες. Οι **αλγόριθμοι επίλυσης** είναι *αποδοτικοί*, η διεπαφή **καλαίσθητη** και *προσαρμοστική*, ενώ η εφαρμογή προάγει την κατανόηση των τεχνικών επίλυσης μέσω **εκπαιδευτικών χαρακτηριστικών**. Ωστόσο, όπως κάθε *καινοτόμο έργο*, υπάρχουν περιθώρια περαιτέρω βελτίωσης και εξέλιξης. Από την **ενίσχυση της αποδοτικότητας του αλγορίθμου** έως την ανάπτυξη *νέων λειτουργιών* και τη βελτίωση της εμπειρίας χρήστη, οι προτάσεις για το μέλλον στοχεύουν στην περαιτέρω ενδυνάμωση της εφαρμογής.

Αυτή η εργασία εξετάζει λεπτομερώς τις *πιθανές βελτιώσεις*, όπως η **τεχνική αναβάθμιση του αλγορίθμου**, η *επέκταση της λειτουργικότητάς του* για την κάλυψη περισσότερων τύπων παζλ, η ενίσχυση της διεπαφής χρήστη με **δυναμικές λειτουργίες** και *προσαρμοστικά γραφικά*, καθώς και η δημιουργία **κοινοτικών χαρακτηριστικών** που θα προάγουν τη διάδραση μεταξύ των χρηστών. Μέσα από αυτές τις βελτιώσεις, η εφαρμογή μπορεί να εξελιχθεί σε ένα **κορυφαίο εργαλείο** τόσο για *διασκέδαση* όσο και για *εκπαίδευση*.

13.4.2 Βελτιώσεις στον Αλγόριθμο Επίλυσης

Η υπάρχουσα υλοποίηση του *αλγορίθμου επίλυσης* είναι **αποδοτική** για ένα μεγάλο εύρος παζλ. Ωστόσο, υπάρχει περιθώριο για **περαιτέρω βελτιώσεις**. Κατά την ανάπτυξη, η *προσεκτική ανάλυση του κώδικα* οδήγησε σε βελτιώσεις που **αύξησαν την απόδοση**. Αυτή η διαδικασία μπορεί να επαναληφθεί, με στόχο την αναγνώριση και την ενίσχυση *συγκεκριμένων τμημάτων* του αλγορίθμου που επηρεάζουν την **αποδοτικότητα** και την *ακρίβεια*.

Ήδη έχουν εντοπιστεί **περιοχές** στις οποίες ο αλγόριθμος μπορεί να βελτιωθεί. Ωστόσο, η *έλλειψη χρόνου* δεν επέτρεψε την εφαρμογή αυτών των βελτιώσεων στο παρόν έργο. Η **εστίαση** σε αυτά τα σημεία στο μέλλον μπορεί να φέρει *ουσιαστικά αποτελέσματα*, βελτιώνοντας περαιτέρω την αποδοτικότητα και τη συνολική απόδοση της εφαρμογής.

13.4.3 Παρουσίαση της Αλγοριθμικής Διαδικασίας

Η *παρουσίαση της προόδου* του αλγορίθμου αποτελεί ένα σημαντικό σημείο για **βελτίωση**. Παρά τις *υπάρχουσες προσπάθειες*, η προβολή των βημάτων επίλυσης δεν είναι αρκετά **λεπτομερής** ώστε να παρέχει στους χρήστες μια πλήρη κατανόηση της *διαδικασίας*.

Μια πιο **οργανωμένη προσέγγιση** στη δομή αυτής της λειτουργίας θα μπορούσε να αναδείξει περισσότερες *πτυχές* της αλγοριθμικής διαδικασίας. Η **ενίσχυση** αυτής της λειτουργίας θα καθιστούσε την εφαρμογή πιο *διαφανή* και θα παρείχε στους χρήστες βαθύτερη **γνώση** για τη λειτουργία της. Με αυτόν τον τρόπο, η εφαρμογή θα μπορούσε να προσφέρει μια πιο *εκπαιδευτική* εμπειρία, ενισχύοντας την κατανόηση των τεχνικών επίλυσης.

13.4.4 Δημιουργία Σελίδας με Πληροφοριακό Υλικό

Η **ενσωμάτωση** μιας σελίδας με *πληροφοριακό υλικό* θα προσέφερε επιπλέον **αξία** στην εφαρμογή. Μια τέτοια σελίδα θα μπορούσε να περιλαμβάνει:

- **Οδηγίες επίλυσης** εικονόσταυρων.
- **Αναλύσεις τεχνικών** και στρατηγικών επίλυσης.
- **Παραδείγματα** από άλλους λύτες.
- **Γενικές πληροφορίες** για την ιστορία και τα είδη των παζλ.

Αυτό το περιεχόμενο θα ήταν ιδιαίτερα χρήσιμο για *νέους χρήστες*, καθώς θα τους βοηθούσε να κατανοήσουν καλύτερα το αντικείμενο. Επιπλέον, θα ενίσχυε τη **χρησιμότητα** της εφαρμογής ως *εκπαιδευτικό εργαλείο*, καθιστώντας την πολύτιμη τόσο για αρχάριους όσο και για προχωρημένους χρήστες.

13.4.5 Βελτίωση Διεπαφής Χρήστη

Η διεπαφή χρήστη της εφαρμογής είναι ήδη *καλαίσθητη* και **λειτουργική**. Έχει σχεδιαστεί ώστε να *προσαρμόζεται αποτελεσματικά* σε διαφορετικά μεγέθη οθόνης, διασφαλίζοντας ότι οι βασικές πληροφορίες παραμένουν ευδιάκριτες. Αυτή η **σχεδιαστική επιλογή** εξασφαλίζει μια *ομαλή εμπειρία χρήστη*, ανεξαρτήτως συσκευής. Παρότι η παρούσα διεπαφή είναι *αξιόλογη*, υπάρχουν περιθώρια για **βελτιώσεις** που θα την καθιστούσαν ακόμα πιο φιλική και ελκυστική.

Μια σημαντική πρόταση είναι η ενσωμάτωση **δυναμικών και προσαρμοστικών γραφικών**. Τέτοια γραφικά θα ενίσχυαν την *οπτική εμπειρία*, διευκολύνοντας την κατανόηση της κατάστασης του παζλ. Για παράδειγμα, η χρήση περισσότερων *χρωμάτων* για την απεικόνιση διαφορετικών καταστάσεων των κελιών θα ήταν ιδιαίτερα χρήσιμη σε πιο σύνθετα σενάρια. Επιπλέον, η δυνατότητα **επιλογής θεμάτων διεπαφής** (*themes*) θα προσέφερε στους χρήστες τη δυνατότητα να προσαρμόζουν την εμφάνιση της εφαρμογής στις προσωπικές τους προτιμήσεις.

Μια άλλη ενδιαφέρουσα προσθήκη θα ήταν η ενσωμάτωση ενός **οδηγού επίλυσης**, ο οποίος θα καθοδηγεί τους χρήστες *βήμα προς βήμα* στην επίλυση πιο σύνθετων παζλ. Ένας τέτοιος οδηγός θα περιλάμβανε λεπτομερείς *επεξηγήσεις*, βοηθώντας τόσο αρχάριους όσο και προχωρημένους χρήστες να κατανοούν καλύτερα τις τεχνικές επίλυσης και να βελτιώνουν τις δεξιότητές τους.

Επιπλέον, η δυνατότητα **προσαρμογής της διάταξης** της διεπαφής στις ανάγκες του χρήστη θα ήταν μια ακόμη χρήσιμη βελτίωση. Για παράδειγμα, οι χρήστες θα μπορούσαν να επιλέγουν συγκεκριμένες λειτουργίες, να αποκρύπτουν λιγότερο σημαντικές πληροφορίες ή να διαμορφώνουν τη διάταξη των στοιχείων σύμφωνα με τις προτιμήσεις τους.

Συνολικά, παρότι η διεπαφή είναι ήδη *καλοσχεδιασμένη*, αυτές οι βελτιώσεις μπορούν να προσφέρουν μια ακόμα πιο **εξατομικευμένη** και *ελκυστική* εμπειρία, καθιστώντας την εφαρμογή πιο *φιλική προς τον χρήστη*.

13.4.6 Ενίσχυση της Διάδρασης με τους Χρήστες

Η *ενσωμάτωση λειτουργιών* που επιτρέπουν στους χρήστες να **δημιουργούν** και να **μοιράζονται** τα δικά τους παζλ μπορεί να βελτιώσει σημαντικά τη διάδραση στην εφαρμογή. Για παράδειγμα, οι χρήστες θα μπορούσαν να *αποθηκεύουν* τα παζλ τους στο **cloud** και να τα *δημοσιεύουν* στον ιστότοπο, ώστε να είναι διαθέσιμα σε άλλους χρήστες.

Αυτή η **δυνατότητα** θα προωθούσε τη *δημιουργικότητα* και θα ενίσχυε την ανάπτυξη μιας *κοινότητας* γύρω από την εφαρμογή. Οι χρήστες θα είχαν τη δυνατότητα να **αλληλεπιδρούν**, να *μοιράζονται ιδέες* και να **συνεργάζονται**, καθιστώντας την εφαρμογή ένα *ζωντανό εργαλείο δημιουργίας* και αλληλεπίδρασης.

13.4.7 Επέκταση της Λειτουργικότητας του Αλγορίθμου

Η *προσθήκη νέων λειτουργιών* στον αλγόριθμο είναι **ζωτικής σημασίας** για την περαιτέρω εξέλιξη της εφαρμογής. Ένα από τα βασικά σημεία για βελτίωση είναι η **ανίχνευση πολλαπλών λύσεων** σε ένα παζλ. Αυτή η λειτουργία μπορεί να παρέχει *κρίσιμες πληροφορίες* για τη φύση του κάθε παζλ, καθιστώντας την εφαρμογή ιδιαίτερα χρήσιμη για **απαιτητικά προβλήματα**.

Επιπλέον, η **ενσωμάτωση νέων τεχνικών επίλυσης** θα διευρύνει τις δυνατότητες του αλγορίθμου. Τέτοιες τεχνικές θα μπορούσαν να επιτρέψουν την επίλυση *πιο πολύπλοκων ή ιδιαίτερων παζλ*, όπως εκείνων με **μοναδικές γεωμετρικές ιδιότητες**, αυξάνοντας τη χρηστικότητα του εργαλείου.

Σημαντική προοπτική για το μέλλον είναι και η **υποστήριξη περισσότερων τύπων παζλ**. Η εφαρμογή μπορεί να επεκταθεί ώστε να περιλαμβάνει *έγχρωμα εικονόσταυρα* ή **τρισδιάστατα παζλ**, τα οποία απαιτούν νέες *τεχνικές επίλυσης* και **καινοτόμες προσεγγίσεις** στην παρουσίαση και τη διαχείριση δεδομένων.

Αυτή η *διεύρυνση του πεδίου εφαρμογής* θα καταστήσει την εφαρμογή ακόμα πιο **ενδιαφέρουσα** και *χρήσιμη* για ένα **ευρύτερο κοινό**, προσελκύοντας νέους χρήστες με *διαφορετικά ενδιαφέροντα*.

13.4.8 Επίλογος

Η εφαρμογή για την επίλυση εικονόσταυρων έχει *επιτυχώς συνδυάσει* **λειτουργικότητα**, **εκπαίδευση** και **ευχρηστία** σε έναν ενιαίο σχεδιασμό που καλύπτει τις ανάγκες τόσο των *αρχάριων* όσο και των *έμπειρων χρηστών*. Οι υπάρχουσες λειτουργίες, όπως η **προσαρμοστική διεπαφή** και η *αποδοτικότητα του αλγορίθμου*, αναδεικνύουν τη **δυναμική** της εφαρμογής, ενώ η *εκπαιδευτική διάσταση* την καθιστά εργαλείο **μοναδικής αξίας**. Παρόλα αυτά, υπάρχουν **σημαντικές ευκαιρίες** για περαιτέρω βελτίωση και εξέλιξη.

Οι προτάσεις που αναλύθηκαν, όπως η *βελτίωση του αλγορίθμου*, η **ενσωμάτωση νέων τύπων παζλ**, η **αναβάθμιση της διεπαφής χρήστη** και η **προσθήκη λειτουργιών κοινοτικής αλληλεπίδρασης**, μπορούν να καθοδηγήσουν την εφαρμογή σε ένα *νέο επίπεδο*. Με στοχευμένες βελτιώσεις, η εφαρμογή μπορεί να προσφέρει **ακόμα πιο καινοτόμες λύσεις**, να καλύψει τις ανάγκες ενός *ευρύτερου κοινού* και να **καθιερωθεί** ως μία από τις **κορυφαίες επιλογές** στην κατηγορία της.

Η *συνέχιση της ανάπτυξης* της εφαρμογής θα απαιτήσει **χρόνο**, **πόρους** και περαιτέρω *ανάλυση*, αλλά οι προοπτικές είναι **ιδιαίτερα ελπιδοφόρες**. Μέσα από αυτή τη διαδικασία, η εφαρμογή μπορεί να εξελιχθεί όχι μόνο ως

εργαλείο επίλυσης, αλλά και ως **πλατφόρμα** που προάγει τη *δημιουργικότητα*, την *εκπαίδευση* και τη *διάδραση*, διασφαλίζοντας τη **μακροπρόθεσμη επιτυχία** και την *αξία* της.

13.5 Επίλογος

Η ανάπτυξη της εφαρμογής για την επίλυση εικονόσταυρων αποτελεί ένα *πολλά υποσχόμενο βήμα* προς την αξιοποίηση της **τεχνολογίας** για την αντιμετώπιση σύνθετων προβλημάτων, συνδυάζοντας **λειτουργικότητα**, *εκπαιδευτική αξία* και **διαδραστικότητα**. Οι υπάρχουσες δυνατότητες, όπως η *αποδοτικότητα του αλγορίθμου*, η **προσαρμογή σε διαφορετικά μεγέθη οθόνης** και η *υποστήριξη εκπαιδευτικών λειτουργιών*, θέτουν ισχυρές βάσεις για περαιτέρω ανάπτυξη.

Οι *προτάσεις βελτίωσης* που εξετάστηκαν, από την **ενίσχυση του αλγορίθμου** έως την **επέκταση της λειτουργικότητας** και την *προσθήκη νέων χαρακτηριστικών στη διεπαφή χρήστη*, αποδεικνύουν ότι υπάρχει **μεγάλος χώρος για εξέλιξη**. Αυτές οι προοπτικές ανοίγουν νέους δρόμους για την αναβάθμιση της εφαρμογής και την καλύτερη ικανοποίηση των αναγκών των χρηστών.

Η εφαρμογή έχει τη *δυνατότητα να εξελιχθεί* σε ένα **ολοκληρωμένο εργαλείο**, το οποίο όχι μόνο καλύπτει τις **ανάγκες των χρηστών**, αλλά προάγει τη *δημιουργικότητα*, την **εκπαίδευση** και τη **διάδραση**. Με *στοχευμένες βελτιώσεις* και επενδύσεις στον σχεδιασμό και την ανάπτυξη, η εφαρμογή μπορεί να **εδραιωθεί** ως *κορυφαία επιλογή* στον τομέα της επίλυσης εικονόσταυρων, παραμένοντας ένα *πολύτιμο εργαλείο* για τη **διεύρυνση γνώσεων** και την ανάπτυξη *δεξιοτήτων* των χρηστών.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] J. Dalgety, “Origins of Cross Reference Grid & Picture Grid Puzzles.”

[2] Nonograms.org, “Methods of solving Japanese crosswords.”

[3] Webpbn.com, “Advanced Puzzle Solving Techniques.”

[4] D. Fedoriaka, “Online Nonogram solver.”

[5] Z. Qi, “NONOGRAM.DEMO.”

[6] K. J. Batenburg and W. A. Kosters, “A Reasoning Framework for Solving Nonograms,” *Unpublished Manuscript*, Unknown. Affiliations: University of Antwerp (Vision Lab), Leiden University (LIACS). Emails: joost.batenburg@ua.ac.be, kosters@liacs.nl.

[7] N. Ueda and T. Nagao, “NP-completeness Results for NONOGRAM via Parsimonious Reductions,” *Department of Computer Science, Tokyo Institute of Technology (Technical Report TR96-0008)*, 1996. c/o Prof. T. Sato, Dept. of Electrical and Electronic Engineering; c/o Prof. O. Watanabe, Dept. of Computer Science; Accessed online: 2023-10-07.

[8] J. Wolter, “The ’pbnsolve’ Paint-by-Number Puzzle Solver.”

[9] J. Wolter, “Survey of Paint-by-Number Puzzle Solvers.”

[10] S. Makrygiannaki, “Nonogram Solver and Designer Repository.”