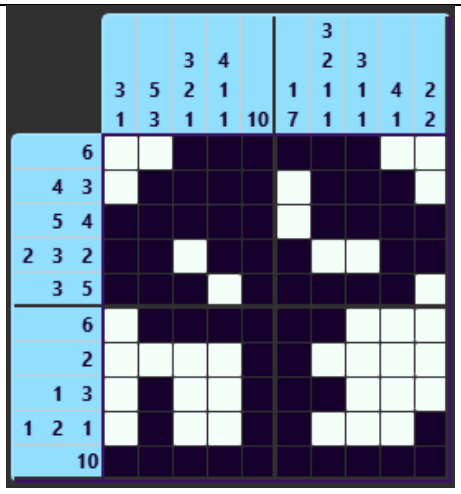
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Δημιουργία Παιχνιδιού παρόμοιου με το Νανόγραμμα»



Του φοιτητή
Κωνσταντίνου Τζουβαλίδη
Αρ. Μητρώου: 134020

Επιβλέπων
Κωνσταντίνος Γουλιάνας
Βαθμίδα ……

Ημερομηνία 31/5/2025

Δημιουργία Παιχνιδιού παρόμοιου με το Νανόγραμμα

Κωδικός Δ.Ε. 25210
Κωνσταντίνος Τζουβαλίδης
Κωνσταντίνος Γουλιάνας
Ημερομηνία ανάληψης Δ.Ε. 25/3/2025
Ημερομηνία περάτωσης Δ.Ε. 31/5/2025

Βεβαιώνω ότι είμαι ο 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.

«Αφιέρωση»

Πρόλογος

Η ενασχόλησή μου με τα λογικά παιχνίδια και η αγάπη μου για την επίλυση γρίφων αποτέλεσαν τους βασικούς λόγους που επέλεξα τη συγκεκριμένη διπλωματική εργασία. Τα nonograms, συνδυάζοντας τη λογική σκέψη με τη δημιουργικότητα, αποτέλεσαν το ιδανικό αντικείμενο μελέτης για να εμβαθύνω τόσο σε τεχνικά θέματα ανάπτυξης λογισμικού όσο και σε ζητήματα σχεδίασης εμπειρίας χρήστη. Μέσα από την υλοποίηση ενός ψηφιακού παιχνιδιού nonogram, είχα την ευκαιρία να εφαρμόσω αλγοριθμικές τεχνικές, να εξετάσω προκλήσεις που σχετίζονται με τη διαχείριση δεδομένων και να βελτιώσω τις δεξιότητές μου στον προγραμματισμό και στον σχεδιασμό διεπαφών. Η εργασία αυτή μου προσέφερε πολύτιμη εμπειρία στη διαχείριση ενός ολοκληρωμένου έργου λογισμικού και συνέβαλε καθοριστικά στην προσωπική και επαγγελματική μου ανάπτυξη.

Περίληψη

Η παρούσα διπλωματική εργασία επικεντρώνεται στην ανάπτυξη ενός ψηφιακού παιχνιδιού Nonogram, ενός λογικού παζλ που συνδυάζει αριθμητικά στοιχεία με τη δημιουργία εικόνας σε μορφή πλέγματος. Η υλοποίηση πραγματοποιήθηκε χρησιμοποιώντας ReactJS, HTML και CSS, με στόχο τη δημιουργία μιας σύγχρονης, αποκριτικής και φιλικής προς τον χρήστη διαδικτυακής εφαρμογής. Ο χρήστης έχει τη δυνατότητα να επιλέξει ένα έτοιμο επίπεδο είτε τυχαία είτε συγκεκριμένα, καθώς και να δημιουργήσει δικά του custom επίπεδα μέσω ενός ενσωματωμένου εργαλείου σχεδίασης. Οι δημιουργίες αποθηκεύονται τοπικά στη συσκευή του χρήστη μέσω IndexedDB, εξασφαλίζοντας αυτονομία και διατήρηση των δεδομένων χωρίς την ανάγκη διακομιστή.

Στην εφαρμογή δίνεται έμφαση στη σωστή απόδοση των κανόνων του παιχνιδιού, όπως ο αυτόματος έλεγχος λύσης, η διαχείριση λαθών και η παροχή βοηθητικών εργαλείων κατά τη διάρκεια της επίλυσης. Παράλληλα, αναπτύχθηκε μηχανισμός για τον χειρισμό της κατάστασης της εφαρμογής και για τη διασφάλιση της ομαλής εμπειρίας του χρήστη ακόμα και σε πολύπλοκα επίπεδα. Η εργασία εξετάζει επίσης προκλήσεις όπως η αποδοτική αποθήκευση δεδομένων σε IndexedDB, η δυναμική ενημέρωση του περιβάλλοντος χρήστη, καθώς και η συνολική αρχιτεκτονική της εφαρμογής με γνώμονα τη μελλοντική επεκτασιμότητα.

Το αποτέλεσμα είναι ένα πλήρως λειτουργικό παιχνίδι Nonogram, το οποίο προσφέρει τόσο προκαθορισμένα όσο και εξατομικευμένα επίπεδα, με έμφαση στη διαδραστικότητα, την ευκολία χρήσης και τη διατήρηση της λογικής πρόκλησης που χαρακτηρίζει το είδος.

«Δημιουργία Παιχνιδιού παρόμοιου με το Νανόγραμμα»

(Design and Development of a Nonogram Game)

«Κωνσταντίνος Τζουβαλίδης»

(Konstantinos Tzouvalidis)

Abstract

This thesis presents the design and development of a web-based application for creating and solving Nonogram (Picross) puzzles. Nonograms are grid-based logic puzzles where users must deduce which cells to fill based on numeric clues. The implemented system provides an interactive environment that supports the construction and real-time solving of black-and-white Nonogram puzzles.

Developed using the ReactJS framework, the application is structured as a Single Page Application (SPA), offering three primary modes: Edit, Play, and Preview. It includes key features such as clue auto-generation, undo/redo functionality, smart hints, automatic X-marking, and progress validation. Data persistence is handled through IndexedDB via the Dexie.js library, enabling complete offline usage without the need for a backend.

Special attention was given to user experience, with responsive grid interaction, visual clues feedback, and support for custom level creation. The logic and architecture were modularized using reusable React components and custom hooks. A hint engine was also implemented, providing context-aware suggestions during gameplay.

The result is a fully functional, standalone puzzle platform that combines clean design, performance, and extensibility. The application is suitable for casual players, puzzle creators, or educational use, and can be further enhanced with features such as colored puzzles, cloud synchronization, and level sharing.

Περιεχόμενα

Κατάλογος Εικόνων

Συντομογραφίες

SPA:	Single Page Application
IDE:	Ολοκληρωμένο Περιβάλλον Ανάπτυξης
Δ.Ε.	Διπλωματική Εργασία

Κεφάλαιο 1ο: Εισαγωγή και Θεωρητικό Υπόβαθρο

1.1 Εισαγωγή

Το παρόν κεφάλαιο λειτουργεί ως εισαγωγή πτυχιακή εργασία και αποσκοπεί στο να παρουσιάσει το γενικότερο πλαίσιο μέσα στο οποίο εκπονείται η εφαρμογή που αναπτύχθηκε. Ξεκινώντας με μια σύντομη αναφορά στην έννοια των Nonograms, παρατίθεται η ιστορική τους διαδρομή και η σημερινή τους μορφή, όπως διαμορφώθηκε μέσα από τεχνολογικές εξελίξεις και τη μαζική υιοθέτησή τους σε ψηφιακά μέσα. Στη συνέχεια, περιγράφονται οι στόχοι και τα παραδοτέα της εργασίας, ενώ γίνεται και μια επισκόπηση των κεφαλαίων που ακολουθούν.

Η πτυχιακή εργασία επικεντρώνεται στην ανάπτυξη μιας διαδικτυακής εφαρμογής επίλυσης και δημιουργίας Nonogram παζλ (ασπρόμαυρων), με στόχο τη χρηστικότητα, την εκπαιδευτική αξία και την αισθητική απλότητα. Παράλληλα, διερευνώνται λειτουργίες που βοηθούν την επίλυση παζλ με έξυπνους αυτοματισμούς (π.χ. hints, auto-x), καθώς και η δυνατότητα δημιουργίας νέων επιπέδων από τον χρήστη. Η εργασία εντάσσεται στο ευρύτερο γνωστικό πεδίο του λογισμικού ανοικτού κώδικα, των λογικών παζλ και της ανάπτυξης διεπαφών χρήστη με ReactJS.

Τέλος, στα επόμενα κεφάλαια της εργασίας αναλύονται διεξοδικά οι διαδικασίες και τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής. Συγκεκριμένα, στο **Κεφάλαιο 2** παρουσιάζεται η μεθοδολογία ανάπτυξης και η αρχιτεκτονική της εφαρμογής, ενώ στο **Κεφάλαιο 3** αναλύονται οι τεχνικές υλοποίησης επιμέρους λειτουργιών όπως τα modes, η αποθήκευση επιπέδων και η λογική επίλυσης. Το **Κεφάλαιο 4** περιλαμβάνει τα συμπεράσματα που προέκυψαν κατά τη διάρκεια της εργασίας, καθώς και προτάσεις βελτίωσης για μελλοντική επέκταση της εφαρμογής. Τέλος, παρατίθενται βιβλιογραφικές αναφορές και παραρτήματα με πρόσθετο υλικό και αποσπάσματα κώδικα όπου κρίνεται απαραίτητο.

1.2 Τι είναι τα Nonograms

Τα Nonogram, γνωστά και ως «εικονογρίφοι», «Japanese puzzles» ή «Picross», αποτελούν είδος λογικού γρίφου που βασίζεται σε ένα πλέγμα κουτιών, το οποίο ο παίκτης καλείται να συμπληρώσει (ή να αφήσει κενό) βάσει αριθμητικών ενδείξεων. Οι αριθμοί που βρίσκονται στις άκρες κάθε γραμμής και στήλης δηλώνουν διαδοχικά σύνολα συνεχόμενων γεμισμένων κελιών. Ο παίκτης, ακολουθώντας αυτές τις ενδείξεις και με τη βοήθεια της λογικής του, προσπαθεί να σχηματίσει την κρυμμένη εικόνα που προκύπτει μέσα από το πλέγμα.

Σε αντίθεση με άλλες μορφές παζλ, τα Nonogram διαθέτουν ένα μοναδικό χαρακτηριστικό: κάθε παζλ έχει μία και μοναδική λύση, την οποία ο παίκτης μπορεί να βρει χωρίς να χρειάζεται να κάνει υποθέσεις. Αυτό τα καθιστά ιδιαίτερα ελκυστικά για άτομα που αγαπούν τις λογικές προκλήσεις.

Οι γρίφοι αυτοί ενισχύουν τη συγκέντρωση, την παρατηρητικότητα και τη λογική σκέψη. Για το λόγο αυτό, έχουν χρησιμοποιηθεί όχι μόνο ως μορφή ψυχαγωγίας, αλλά και σε εκπαιδευτικά και νοητικά προγράμματα βελτίωσης δεξιοτήτων, σε σχολεία και σε περιβάλλοντα νοητικής αποκατάστασης.

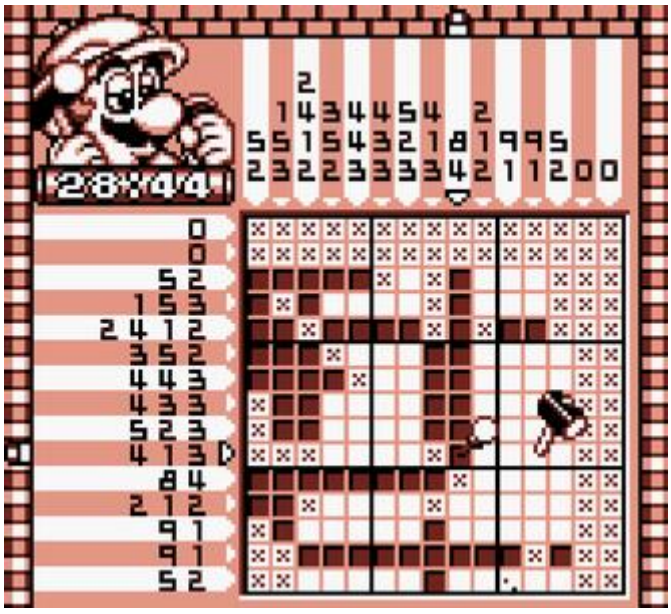
1.3 Ιστορική Αναδρομή και Παραδείγματα

Τα Nonograms ξεκίνησαν την πορεία τους στη δεκαετία του 1980 στην Ιαπωνία. Το 1987, η σχεδιάστρια παζλ Non Ishida δημιούργησε έναν διαγωνισμό στον οποίο οι συμμετέχοντες έπρεπε να σχεδιάσουν εικόνες με φώτα παραθύρων σε ουρανοξύστες. Από την ιδέα αυτή προέκυψαν τα πρώτα Nonograms, τα οποία δημοσιεύτηκαν αργότερα σε ιαπωνικά περιοδικά γρίφων. Παράλληλα, ο Tetsuya Nishio, βοήθησε να καθιερωθεί το συγκεκριμένο είδος παζλ με την ονομασία «Paint by Numbers».

Στις αρχές της δεκαετίας του 1990, τα Nonograms πέρασαν στην Ευρώπη. Η The Telegraph στο Ηνωμένο Βασίλειο δημοσίευσε καθημερινά Nonograms με την ονομασία "Griddlers", ενώ το είδος έγινε επίσης γνωστό και ως "Pic-a-Pix", "Hanjie" και "Japanese Crosswords". Αυτή η πληθώρα ονομασιών αντανακλά τη διεθνή αναγνώριση αλλά και τον συνεχιζόμενο πειραματισμό με τον τρόπο διάδοσης του παζλ.

Η μαζική τους διάδοση, ωστόσο, ήρθε με την είσοδό τους στον χώρο των ηλεκτρονικών παιχνιδιών. Η Nintendo υπήρξε πρωτοπόρος στη μεταφορά των Nonograms στο ψηφιακό περιβάλλον, ξεκινώντας με τη σειρά Picross στο Game Boy το 1995, με τίτλο "Mario's Picross". Ο παίκτης αναλάμβανε τον ρόλο του Mario, ο οποίος έπρεπε να επιλύσει Nonogram παζλ για να αποκαλύψει εικόνες. Αν και το παιχνίδι δεν γνώρισε άμεση επιτυχία στην Αμερική, σημείωσε μεγάλη επιτυχία στην Ιαπωνία και απέκτησε αφοσιωμένο κοινό.

- Η Nintendo συνέχισε τη στήριξη της σειράς με τίτλους όπως:
- Mario's Super Picross (Super Famicom – μόνο Ιαπωνία)
- Picross DS (2007 – Nintendo DS)
- Picross 3D και Picross 3D: Round 2 (Nintendo DS και 3DS)
- Η πιο πρόσφατη σειρά, Picross S, διατίθεται για το Nintendo Switch με συνεχείς κυκλοφορίες (S1 έως S9+).



Εικόνα 1.1: Mario’s Picross, το πρώτο Nonogram ηλεκτρονικό παιχνίδι

Παράλληλα, πολλές ανεξάρτητες πλατφόρμες (indie developers) δημιούργησαν Nonogram-based παιχνίδια για PC, Android και iOS, όπως τα:

- Pixel Puzzle Collection (Konami, iOS/Android)
- Logic Pic (σε Android)
- Paint it Back (Steam και iOS)
- Nonograms Katana (Android/iOS και Web)

Η μετάβαση από έντυπο σε ψηφιακή μορφή άνοιξε νέες προοπτικές, δίνοντας τη δυνατότητα στους χρήστες να δημιουργούν, να διαμοιράζονται και να επιλύουν Nonograms με μεγαλύτερη ευκολία. Σήμερα, πλατφόρμες όπως τα nonograms.org, onlinenonograms.com, griddlers.net και πολλά ακόμη, φιλοξενούν χιλιάδες παζλ, που συνεχώς αυξάνονται μέσω της συνεισφοράς της κοινότητας.

Η μακρόχρονη παρουσία των Nonograms στην ιστορία των λογικών παζλ αποδεικνύει τη διαχρονική τους αξία. Πρόκειται για μια μορφή "παιχνιδιού-πρόκλησης" που όχι μόνο προάγει την κριτική σκέψη, αλλά προσφέρει και αισθητική ικανοποίηση στον παίκτη μέσω της αποκάλυψης της τελικής εικόνας.

1.4 Περιγραφή του Αντικειμένου της Εφαρμογής

Η εφαρμογή που αναπτύχθηκε στα πλαίσια της παρούσας πτυχιακής εργασίας έχει ως βασικό στόχο την παροχή ενός διαδραστικού περιβάλλοντος σχεδίασης και επίλυσης Nonogram γρίφων, ακολουθώντας το πρότυπο διεπαφής ιστοσελίδων όπως το nonograms.org και το onlinenonograms.com. Ο σχεδιασμός της βασίζεται σε σύγχρονες πρακτικές αλληλεπίδρασης και εργονομίας, ενώ δίνει έμφαση στη λειτουργικότητα, στην ακρίβεια και στην εμπειρία του τελικού χρήστη. Παρέχει ένα σύνολο εργαλείων και λειτουργιών που καθιστούν δυνατή την αλληλεπίδραση με το πλέγμα του γρίφου, είτε για την επίλυσή του, είτε για τον σχεδιασμό νέων επιπέδων.

Η εφαρμογή περιλαμβάνει τρεις βασικές λειτουργίες (modes), τις οποίες μπορεί να ενεργοποιήσει ο χρήστης μέσω του μενού ελέγχου. Παρακάτω περιγράφονται αναλυτικά.

1.4.1 Λειτουργία Σχεδίασης (Edit Mode)

Η λειτουργία σχεδίασης επιτρέπει στον χρήστη να δημιουργήσει έναν νέο Nonogram γρίφο από την αρχή. Ο χρήστης μπορεί να καθορίσει το μέγεθος του πλέγματος (πλάτος και ύψος) και να σχεδιάσει την εικόνα που επιθυμεί γεμίζοντας ή αδειάζοντας κελιά. Η εφαρμογή υπολογίζει αυτόματα τις αριθμητικές ενδείξεις (clues) για κάθε γραμμή και στήλη, βάσει των γεμισμένων κελιών.

Οι βασικές δυνατότητες της λειτουργίας σχεδίασης περιλαμβάνουν:

- Καθορισμό των διαστάσεων του επιπέδου.
- Διαδραστική σχεδίαση με χρήση του ποντικιού.
- Αυτόματη παραγωγή των ενδείξεων (clues).
- Αποθήκευση του επιπέδου με όνομα επιλογής.
- Δυνατότητα εισαγωγής έτοιμων επιπέδων

Η λειτουργία αυτή απευθύνεται τόσο σε προχωρημένους χρήστες που θέλουν να δημιουργήσουν σύνθετους γρίφους, όσο και σε αρχάριους που θέλουν να πειραματιστούν με το είδος των Nonogram.

1.4.2 Λειτουργία Παιχνιδιού (Play Mode)

Στη λειτουργία παιχνιδιού, ο χρήστης μπορεί να επιλέξει και να επιχειρήσει να λύσει έναν αποθηκευμένο ή προκαθορισμένο Nonogram γρίφο. Ο σκοπός είναι η αποκάλυψη της κρυμμένης εικόνας μέσα από λογική και παρατήρηση των αριθμητικών ενδείξεων.

Η λειτουργία προσφέρει τα παρακάτω εργαλεία και δυνατότητες:

- **Επιλογή τρόπου σχεδίασης**: Συμπλήρωση κελιών με χρώμα (γεμισμένα), με τελείες (σημειώσεις), ή με εναλλαγή καταστάσεων (αυτόματη λειτουργία).
- **Auto-X**: Αυτόματη συμπλήρωση «Χ» σε κελιά που δεν ανήκουν σε καμία ένδειξη, μόλις ολοκληρωθεί σωστά μια γραμμή ή στήλη.
- **Auto clue marking**: Αυτόματη επισήμανση των clues που έχουν ολοκληρωθεί επιτυχώς.
- **Undo/Redo:** Δυνατότητα αναίρεσης ή επαναφοράς προηγούμενων κινήσεων χωρίς περιορισμούς.
- **Υποδείξεις (Hints):** Με το πάτημα ενός κουμπιού, ο χρήστης μπορεί να λάβει μία λογικά επιλεγμένη βοήθεια για να συνεχίσει τη λύση.
- **Έλεγχος λύσης**: Η εφαρμογή διαθέτει ειδική λειτουργία επαλήθευσης της λύσης με βάση την αυθεντική απάντηση.

Η λειτουργία παιχνιδιού έχει σχεδιαστεί έτσι ώστε να υποστηρίζει τόσο την εξερεύνηση όσο και την πρόκληση, παρέχοντας βοήθεια όταν χρειάζεται αλλά χωρίς να αφαιρεί τον βασικό χαρακτήρα επίλυσης μέσω λογικής.

1.4.3 Λειτουργία Προεπισκόπησης (Preview Mode)

Η λειτουργία προεπισκόπησης επιτρέπει την άμεση εναλλαγή της προβολής από την κατάσταση παιχνιδιού στην πλήρη εμφάνιση της τελικής εικόνας. Ο χρήστης μπορεί να δει τη λύση του επιπέδου, είτε για να επαληθεύσει τη δική του προσέγγιση, είτε για να μελετήσει τον σχεδιασμό ενός νέου γρίφου. Η προεπισκόπηση δεν επηρεάζει την πρόοδο του παίκτη, καθώς η κατάσταση επιστρέφει εύκολα στην προηγούμενη με την απενεργοποίηση της λειτουργίας.

1.4.4 Συνοπτική Παρουσίαση Δυνατοτήτων

Η εφαρμογή, σε συνολική αποτίμηση, παρέχει:

- Πλήρη περιβάλλον σχεδίασης και επίλυσης Nonogram.
- Εύχρηστο γραφικό περιβάλλον με διαχωρισμό ανά 5 γραμμές/στήλες για καλύτερη κατανόηση.
- Αυτόματη παραγωγή και επισήμανση ενδείξεων.
- Λειτουργίες υποβοήθησης όπως undo, redo, hints και auto-X.
- Εμφάνιση της λύσης με την προεπισκόπηση.
- Μηχανισμό επαλήθευσης για την ορθή ολοκλήρωση του παζλ.

- Βάση επιπέδων που μπορούν να αποθηκευτούν, φορτωθούν και επαναχρησιμοποιηθούν.
- Εισαγωγή έτοιμων επιπέδων μέσω αρχείων .xml.

Η αρχιτεκτονική της εφαρμογής έχει σχεδιαστεί έτσι ώστε να μπορεί να επεκταθεί στο μέλλον με νέες δυνατότητες, όπως υποστήριξη για έγχρωμους γρίφους, εισαγωγή εικόνας για αυτόματη παραγωγή Nonogram, καθώς και μεταφορά της σε περιβάλλον Android.

1.5 Τεχνολογίες Ανάπτυξης και Εργαλεία Υλοποίησης

Η ανάπτυξη της παρούσας εφαρμογής πραγματοποιήθηκε με τη χρήση σύγχρονων τεχνολογιών και εργαλείων που ανταποκρίνονται στις απαιτήσεις της δημιουργίας ενός διαδραστικού και φιλικού προς τον χρήστη περιβάλλοντος. Η επιλογή των τεχνολογιών έγινε με γνώμονα τη σταθερότητα, την επεκτασιμότητα, αλλά και τη διευκόλυνση της ανάπτυξης γραφικού περιβάλλοντος σε περιβάλλον ιστού (web).

Βασική τεχνολογία ανάπτυξης αποτελεί η JavaScript, με την αξιοποίηση της βιβλιοθήκης ReactJS. Η React είναι ιδιαίτερα διαδεδομένη για τη δημιουργία διαδραστικών και αποδοτικών web εφαρμογών, προσφέροντας ταυτόχρονα υψηλό βαθμό επαναχρησιμοποίησης κώδικα. Η χρήση της επιτρέπει τη διαχείριση πολύπλοκων καταστάσεων (state management) με τρόπο απλό και κατανοητό.

Η μορφοποίηση του γραφικού περιβάλλοντος υλοποιήθηκε με τη χρήση CSS (Cascading Style Sheets), ώστε να δημιουργηθεί ένα καλαίσθητο και λειτουργικό περιβάλλον εργασίας για τον τελικό χρήστη. Επιπλέον, αξιοποιήθηκαν σύγχρονες τεχνικές, όπως CSS μεταβάσεις και animation, για την οπτική ανατροφοδότηση κατά την εκτέλεση ενεργειών του χρήστη, π.χ. κατά τη λήψη υποδείξεων (hints).

Για τη διαχείριση δεδομένων και αποθήκευση επιπέδων (nonogram puzzles), χρησιμοποιήθηκε η τεχνολογία IndexedDB μέσω μιας βιβλιοθήκης αποθήκευσης (Dexie.JS), με στόχο την αποθήκευση τοπικών δεδομένων εντός του προγράμματος περιήγησης, χωρίς ανάγκη σύνδεσης σε απομακρυσμένο διακομιστή.

Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε το WebStorm της JetBrains, ένα ευρέως διαδεδομένο και Ολοκληρομένο Περιβάλλον Ανάπτυξης (IDE), το οποίο υποστηρίζει μια πληθώρα επεκτάσεων και εργαλείων που διευκολύνουν την ανάλυση, τη συγγραφή και τη δοκιμή του κώδικα.

Η επιλογή των παραπάνω τεχνολογιών και εργαλείων συνέβαλε καθοριστικά στην ολοκλήρωση μιας λειτουργικής, επεκτάσιμης και εύχρηστης εφαρμογής, που ικανοποιεί τις απαιτήσεις του θέματος της πτυχιακής εργασίας.

1.6 Επίλογος

Στο παρόν κεφάλαιο παρουσιάστηκε το γενικό πλαίσιο της διπλωματικής εργασίας, με στόχο την εισαγωγή στον κόσμο των νανογραμμάτων και την τεχνολογική υλοποίηση ενός σύγχρονου ψηφιακού περιβάλλοντος επίλυσης και δημιουργίας τους. Αρχικά, αναλύθηκε η προέλευση και η εξέλιξη των

nonograms ως είδος παζλ λογικής, ενώ δόθηκε έμφαση στην παρουσία τους σε ηλεκτρονικά παιχνίδια, όπως οι τίτλοι της Nintendo, και στη σταδιακή τους ενσωμάτωση σε εφαρμογές λογισμικού.

Στη συνέχεια, έγινε περιγραφή των λειτουργιών που προσφέρει η εφαρμογή που αναπτύχθηκε, όπως τα τρία διαφορετικά modes (Play, Edit και Preview), η παροχή βοηθητικών εργαλείων επίλυσης (hint, αυτόματη συμπλήρωση, undo-redo, έλεγχος ορθότητας), καθώς και η δυνατότητα σχεδίασης νέων επιπέδων ή χρήσης υπαρχόντων. Αναφέρθηκαν επίσης οι τεχνολογίες και τα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής, όπως η βιβλιοθήκη ReactJS και η γλώσσα JavaScript.

Το Κεφάλαιο 1 έθεσε τις βάσεις για την κατανόηση του αντικειμένου και των στόχων της εργασίας, ενώ στα επόμενα κεφάλαια θα παρουσιαστεί αναλυτικά η μεθοδολογία ανάπτυξης, η δομή του συστήματος, τα τεχνικά χαρακτηριστικά της εφαρμογής καθώς και η αξιολόγησή της ως εργαλείο εκπαιδευτικού και ψυχαγωγικού χαρακτήρα.

Κεφάλαιο 2ο: Αρχιτεκτονική και Υλοποίηση της Εφαρμογής

2.1 Εισαγωγή

Στο παρόν κεφάλαιο αναλύεται σε βάθος η δομή, η τεχνολογική υλοποίηση και η αρχιτεκτονική της διαδικτυακής εφαρμογής Nonogram. Η εφαρμογή σχεδιάστηκε με στόχο την παροχή ενός εργονομικού, αξιόπιστου και αποδοτικού περιβάλλοντος επίλυσης και δημιουργίας λογικών γρίφων τύπου Nonogram. Η ανάλυση καλύπτει τα επιμέρους τεχνικά στοιχεία του συστήματος, τις τεχνολογίες που επιλέχθηκαν, τη ροή των λειτουργιών από την πλευρά του χρήστη, τις σχεδιαστικές αποφάσεις, καθώς και τις προκλήσεις που αντιμετωπίστηκαν κατά την ανάπτυξη του έργου.

2.2 Απαιτήσεις Εφαρμογής

Οι απαιτήσεις της εφαρμογής Nonogram διακρίνονται σε δύο βασικές κατηγορίες: λειτουργικές και μη λειτουργικές. Οι πρώτες καθορίζουν τις δυνατότητες που οφείλει να προσφέρει το σύστημα στον τελικό χρήστη, ενώ οι δεύτερες σχετίζονται με τα ποιοτικά χαρακτηριστικά της εφαρμογής, όπως η απόδοση, η αξιοπιστία και η χρηστικότητα.

Λειτουργικές απαιτήσεις:

- Υλοποίηση πλήρως λειτουργικού περιβάλλοντος για την επίλυση γρίφων Nonogram.
- Παροχή εργαλείου δημιουργίας νέων επιπέδων μέσω διαδραστικής σχεδίασης.
- Αυτόματος υπολογισμός αριθμητικών ενδείξεων (clues) για κάθε νέα δημιουργία.
- Υποστήριξη τριών λειτουργικών καταστάσεων: Edit, Play και Preview.
- Παροχή υποβοηθητικών εργαλείων όπως undo, redo, hints και auto-X.
- Δυνατότητα αποθήκευσης και ανάκτησης επιπέδων μέσω IndexedDB.
- Υποστήριξη εισαγωγής έτοιμων επιπέδων μέσω αρχείων

Μη λειτουργικές απαιτήσεις:

- Άμεση απόκριση της διεπαφής χρήστη κατά την αλληλεπίδραση με το πλέγμα.
- Λειτουργία χωρίς ανάγκη σύνδεσης σε απομακρυσμένο διακομιστή (local-first αρχιτεκτονική).
- Εργονομικός σχεδιασμός διεπαφής και αισθητικά ευχάριστο περιβάλλον χρήστη.
- Δομή αρχιτεκτονικής που να επιτρέπει μελλοντική επέκταση, π.χ. για έγχρωμους γρίφους.

Η ικανοποίηση των παραπάνω απαιτήσεων αποτέλεσε καθοριστικό παράγοντα για τον σχεδιασμό και την υλοποίηση της εφαρμογής, με στόχο τη δημιουργία ενός σταθερού και ολοκληρωμένου ψηφιακού εργαλείου ψυχαγωγίας και εκπαίδευσης.

2.3 Λειτουργικές καταστάσεις (Modes)

Η εφαρμογή Nonogram έχει σχεδιαστεί ώστε να υποστηρίζει πολλαπλά σενάρια χρήσης, προσφέροντας στον τελικό χρήστη τη δυνατότητα να δημιουργήσει, να επεξεργαστεί και να επιλύσει γρίφους λογικής σε ένα πλήρως διαδραστικό περιβάλλον. Στην παρούσα ενότητα παρουσιάζονται οι χαρακτηριστικές λειτουργικές καταστάσεις (Modes), με στόχο την απεικόνιση της λειτουργικής συμπεριφοράς του συστήματος υπό διαφορετικά σενάρια.

Play Mode: Επίλυση υπάρχοντος Nonogram

Ο χρήστης εισέρχεται στη λειτουργία Play όταν επιλέξει έναν διαθέσιμο γρίφο από τη λίστα αποθηκευμένων επιπέδων στο πλαινό μενού (Sidebar) ή πατώντας το Create Game (Random), για να διαλέξει ένα τυχαίο επίπεδο. Η διεπαφή εμφανίζει το πλέγμα και τις αριθμητικές ενδείξεις (clues) στις άκρες των γραμμών και στηλών. Ο χρήστης αρχίζει την επίλυση, χρησιμοποιώντας το ποντίκι για να γεμίσει ή να σημειώσει κελιά. Καθώς προχωρά, έχει πρόσβαση σε λειτουργίες όπως auto-X, undo/redo και υποδείξεις (hints). Μετά την ολοκλήρωση, ενεργοποιεί τη λειτουργία επαλήθευσης για να διαπιστώσει αν η λύση είναι σωστή.

Edit Mode: Δημιουργία νέου επιπέδου

Ο χρήστης επιλέγει τη λειτουργία Edit. Ορίζει τις διαστάσεις του πλέγματος (π.χ. 10x10) και ξεκινά να σχεδιάζει την εικόνα γεμίζοντας συγκεκριμένα κελιά. Επιπλέον, μπορεί να διαλέξει να εισάγει (Import) ένα έτοιμο πλέγμα από αρχείο σε μορφή XML που ακολουθεί συγκεκριμένη μορφή (WebPBN format). Με οποιοδήποτε τρόπο διάλεξε ο χρήστης να γεμίσει τα κελιά, το σύστημα υπολογίζει αυτόματα τις αριθμητικές ενδείξεις για κάθε γραμμή και στήλη. Ο χρήστης μπορεί να αποθηκεύσει το επίπεδο τοπικά, δίνοντάς του ένα όνομα. Το αποθηκευμένο επίπεδο μπορεί στη συνέχεια να επανεμφανιστεί για περαιτέρω επεξεργασία ή επίλυση.

Preview Mode: Προεπισκόπηση τελικής εικόνας

Κατά τη διάρκεια της δημιουργίας ή επίλυσης ενός γρίφου, ο χρήστης μπορεί να ενεργοποιήσει τη λειτουργία Preview. Σε αυτή την περίπτωση, εμφανίζεται άμεσα η τελική εικόνα που δημιουργείται από τη διάταξη των γεμισμένων κελιών. Η λειτουργία αυτή είναι ιδιαίτερα χρήσιμη για τον δημιουργό του επιπέδου, καθώς επιτρέπει την επαλήθευση του σχεδίου πριν την οριστική αποθήκευση.

Οι παραπάνω ροές χρήσης καλύπτουν τα βασικά σενάρια αλληλεπίδρασης με το σύστημα. Η σχεδίαση της εφαρμογής εξασφαλίζει ότι οι λειτουργίες αυτές υλοποιούνται με συνέπεια και σαφήνεια, προάγοντας τη θετική εμπειρία του χρήστη.

2.4 Δομή, επεκτασιμότητα και απόδοση

Η τεχνολογική υποδομή της εφαρμογής Nonogram σχεδιάστηκε ώστε να καλύπτει πλήρως τις απαιτήσεις διαδραστικότητας, απόκρισης και ανεξαρτησίας από backend υποδομή. Για τον σκοπό αυτό αξιοποιήθηκε ένα σύνολο τεχνολογιών του οικοσυστήματος JavaScript που υποστηρίζουν δυναμικές single-page εφαρμογές. Η επιλογή των εργαλείων έγινε με βάση την κοινή χρήση τους στη βιομηχανία, την τεκμηρίωση, την απόδοση και την ευκολία εκμάθησης και συντήρησης.

ReactJS

Η βιβλιοθήκη ReactJS αποτέλεσε τη βάση για την κατασκευή της διεπαφής χρήστη. Η React είναι μία declarative, component-based βιβλιοθήκη JavaScript που επιτρέπει την κατασκευή ευέλικτων, δυναμικών και γρήγορων διεπαφών με υψηλό βαθμό επαναχρησιμοποίησης και ορθής διαχείρισης κατάστασης [6]. Το βασικό πλεονέκτημα της React είναι η λειτουργία του virtual DOM, ο οποίος βελτιώνει δραστικά την απόδοση. Αντί για πλήρη ανανέωση του DOM, η React συγκρίνει (diffing) την προηγούμενη και την τρέχουσα κατάσταση της εφαρμογής και εφαρμόζει μόνο τις απαραίτητες αλλαγές, μειώνοντας τον χρόνο απόκρισης.

Η React βασίζεται σε λειτουργικά components, καθένα από τα οποία έχει σαφή ευθύνη. Για παράδειγμα:

- Το index.js ορίζει τη βασική δομή της εφαρμογής και διαχειρίζεται την αρχικοποίηση.
- Το Game.js είναι υπεύθυνο για τη διαχείριση του παιχνιδιού, των modes και της κατάστασης.
- Το NonogramTable.js χειρίζεται το rendering του πλέγματος και την οπτική ανατροφοδότηση.

Η modular αρχιτεκτονική των components επιτρέπει:

- Εύκολο εντοπισμό και απομόνωση σφαλμάτων.
- Ανεξάρτητη ανάπτυξη και testing κάθε μέρους.
- Δυνατότητα αντικατάστασης ή βελτίωσης επιμέρους λειτουργιών χωρίς επηρεασμό του υπόλοιπου συστήματος.

Για την εσωτερική διαχείριση της κατάστασης, αξιοποιήθηκαν React Hooks (useState, useEffect) και όπου χρειαζόταν κοινή κατάσταση, έγινε χρήση του React Context API (MainContext.js). Το context επιτρέπει σε απομακρυσμένα components να έχουν πρόσβαση σε shared πληροφορίες, χωρίς την ανάγκη prop drilling.

Η ενσωμάτωση εξωτερικών βιβλιοθηκών γίνεται απρόσκοπτα, όπως για παράδειγμα του ReactCardFlip για την εναλλαγή μεταξύ Play/Preview, και της Dexie για IndexedDB storage. Η React διαθέτει επίσης πλούσια τεκμηρίωση και debugging εργαλεία (π.χ. React DevTools), διευκολύνοντας την ανάπτυξη και την επίλυση προβλημάτων.

Η επιλογή της React έγινε έναντι εναλλακτικών όπως:

- **Vue.js**: Παρότι επίσης component-based, δεν διαθέτει τη σταθερότητα και τον βαθμό ελευθερίας της React.
- **Angular**: Ένα πλήρες framework που θεωρήθηκε πιο βαρύ και περίπλοκο για τις ανάγκες μιας μικρής SPA.

Dexie.js και IndexedDB – Αποθήκευση χωρίς backend

Η αποθήκευση των επιπέδων Nonogram επιτυγχάνεται με χρήση της **IndexedDB**, μιας εγγενώς υποστηριζόμενης βάσης δεδομένων στο περιβάλλον του browser [7]. Είναι προτιμότερη έναντι άλλων client-side αποθηκευτικών λύσεων (όπως localStorage) για τους εξής λόγους:

- Υποστηρίζει σύνθετες δομές δεδομένων (αντικείμενα).
- Είναι ασύγχρονη, αποφεύγοντας μπλοκαρίσματα του UI.
- Υποστηρίζει indexing και range-based queries.
- Μπορεί να αποθηκεύσει μεγάλα δεδομένα (δεκάδες MBs).

Η Dexie.js χρησιμοποιείται ως wrapper πάνω από την IndexedDB API. Απλοποιεί τη σύνταξη, διευκολύνει την υλοποίηση με transactions, schema και versioning και προσφέρει async υποστήριξη μέσω promises. Το context MainContext.js αρχικοποιεί την Dexie και παρέχει πρόσβαση στο επίπεδο δεδομένων (dbLevels) σε όλη την εφαρμογή.

Παράδειγμα δημιουργίας βάσης:

```
const db = new Dexie("nonogram");
db.version( versionNumber: 1).stores( schema: {
    dbLevels: "++id,name,cells,width,height"
});
```

Οι πίνακες επιπέδων περιλαμβάνουν metadata, clues και grid data ως JSON strings, επιτρέποντας:

- Τοπική αποθήκευση custom επιπέδων.
- Αναζήτηση με βάση τίτλο ή ημερομηνία.
- Υποστήριξη πολλαπλών επιπέδων και preview.

CSS3 και HTML5 – Layout, animation και responsiveness

Το UI σχεδιάστηκε με βάση **CSS Grid** και **Flexbox**, επιτρέποντας τη δυναμική διάταξη του πλέγματος και των clues.

- Τα πλέγματα χωρίζονται ανά 5 κελιά με border styling (nth-child).
- Το layout είναι responsive για υποστήριξη οθονών κινητών/desktop.
- Χρησιμοποιούνται CSS μεταβλητές (--cell-size, --color-correct) για δυναμική αλλαγή θέματος ή λειτουργίας.

Με τη χρήση CSS transitions, προσφέρεται οπτικό feedback:

- Αλλαγή χρώματος κατά το hover ή click.
- Εμφάνιση βοηθητικών στοιχείων με animation (fade-in, scale).

Σε επίπεδο HTML5, αξιοποιούνται semantic tags όπως:

- **<main>:** για το βασικό περιεχόμενο.
- **<button>:** με role attributes για προσβασιμότητα.
- **<section>:** για διαχωρισμό θεματικών περιοχών (toolbar, puzzle area, status).

Περιβάλλον ανάπτυξης και debugging εργαλεία

Η συγγραφή και δοκιμή του κώδικα πραγματοποιήθηκε σε **WebStorm**, IDE που παρέχει:

- Live linting με ESLint.
- JSX/JS υποστήριξη.
- Refactoring και debugger.

Κατά τη φάση debugging χρησιμοποιήθηκαν:

- **React Developer Tools**: Για παρακολούθηση components, props και hooks.
- **Chrome DevTools – Performance Tab**: Για μέτρηση re-rendering χρόνων.
- **Console-based debugging**: Με προσωρινά logs σε κρίσιμα σημεία (console.log(gridState)).

Η ανάπτυξη ακολούθησε οργανωμένη δομή φακέλων (components/, utils/, hooks/, storage/), με σαφή διαχωρισμό ευθυνών ανά module.

Συνολική αξιολόγηση και συμβολή

Η τεχνολογική υποδομή της εφαρμογής εξυπηρετεί:

1. Την **πλήρη offline λειτουργικότητα**, χωρίς server.
2. Τη **διαχείριση πολύπλοκης κατάστασης** (undo/redo, hints).
3. Τη **σαφή, εργονομική διεπαφή**, η οποία υποστηρίζει τις λειτουργικές καταστάσεις (βλ. 2.3).
4. Την **ευκολία μελλοντικής επέκτασης**, π.χ. για multiplayer υποστήριξη, σύνδεση με cloud αποθήκευση, ή έγχρωμους γρίφους.

Η επιλογή τεχνολογιών δεν έγινε απλώς βάσει δημοτικότητας, αλλά με γνώμονα την καταλληλότητά τους για εφαρμογές puzzle-λογικής, οι οποίες απαιτούν ακρίβεια, σταθερότητα και απρόσκοπτη αλληλεπίδραση με τον χρήστη.

2.5 Ανάλυση αρχιτεκτονικής

Η αρχιτεκτονική της εφαρμογής Nonogram στηρίζεται πλήρως στο ReactJS και δομείται με γνώμονα την επεκτασιμότητα, την ευκολία συντήρησης και την σαφή διάκριση αρμοδιοτήτων ανά component. Βασικός στόχος του αρχιτεκτονικού σχεδιασμού ήταν η δημιουργία ενός περιβάλλοντος όπου η λογική επίλυσης, η αλληλεπίδραση του χρήστη και η απεικόνιση της πληροφορίας να διατηρούνται όσο το δυνατόν απομονωμένα.

Ο κορμός της εφαρμογής εστιάζεται στο component Game.js, το οποίο ελέγχει τη γενική ροή λειτουργίας και τη διαχείριση των modes (Edit, Play, Preview). Περιλαμβάνει όλη τη βασική λογική της εφαρμογής, συμπεριλαμβανομένων των μεθόδων: δημιουργίας νέου πλέγματος (createEmptyGrid, createEmptyPlayingGrid), χειρισμού επιπέδων (handleSaveLevelClick, handleResetClick), ενεργοποίησης υποδείξεων (handleHintClick), ελέγχου λύσης (handleCheckSolvedClick) εισαγωγής έτοιμων επιπέδων (handleImportLevel) και χειρισμού πλοήγησης (handlePreviewClick).

Το πλέγμα Nonogram αποδίδεται μέσω του component NonogramTable.js, το οποίο αποτελεί το βασικό οπτικό interface του χρήστη. Αναλαμβάνει την απόδοση των κελιών, των αριθμητικών ενδείξεων

(clues) και των δυναμικών αλληλεπιδράσεων. Η λειτουργικότητα περιλαμβάνει πλήρεις μηχανισμούς καταγραφής και απόκρισης σε ενέργειες του χρήστη, όπως κλικ, drag, session-based σχεδίαση και μεταβολή καταστάσεων (π.χ. DOT, FILLED, X). Οι συναρτήσεις handleMouseDown, handleMouseEnter και fillCell ρυθμίζουν την τροποποίηση της κατάστασης κελιών, ενώ οι markCluesAsDone και applyAutoX εμπλέκονται στην λογική των clues.

Το component αυτό διατηρεί τοπικές καταστάσεις (π.χ. sessionCells, drawAction) που συνδυάζονται με τα props του ανώτερου Game.js (όπως hintCells, drawMode, autoMarkClues) για τον πλήρη έλεγχο της συμπεριφοράς του grid. Η επικοινωνία μεταξύ χρήστη και grid γίνεται με πλήρως controlled τρόπο, με τα δεδομένα να ρέουν από το Game.js προς το NonogramTable και πίσω μέσω callbacks (setCells, onCellClick).

Η διαχείριση του ιστορικού κινήσεων (undo/redo) επιτυγχάνεται μέσω custom React hook useUndoRedo, το οποίο διαχειρίζεται τις καταστάσεις playingCells, undoStack, redoStack και παρέχει τις συναρτήσεις setPlayingCellsWithHistory, handleUndo, handleRedo. Η προσέγγιση είναι modular, με τον μηχανισμό να είναι ανεξάρτητος από την υπόλοιπη λογική και εύκολα επαναχρησιμοποιήσιμος.

Η αποθήκευση των επιπέδων επιτυγχάνεται μέσω του context MainContext.js, το οποίο παρέχει πρόσβαση στο IndexedDB (Dexie), καθώς και σε μεταβλητές όπως selectedLevel και size. Η χρήση του context εξασφαλίζει ότι βασικά δεδομένα είναι προσβάσιμα σε όλα τα components της εφαρμογής χωρίς prop drilling, ενισχύοντας τη σαφήνεια και μειώνοντας την πολυπλοκότητα.

Η αρχιτεκτονική του έργου ακολουθεί τον εξής επιμερισμό βασικών components:

App.js: Αποτελεί το entry point της εφαρμογής. Φορτώνει τα δεδομένα, διαχειρίζεται την αλλαγή μεταξύ λειτουργιών (Play, Edit, Preview) και περνά την κατάσταση στο Game component. Δεν έχει δική του λογική, αλλά λειτουργεί ως "container" για το interface.

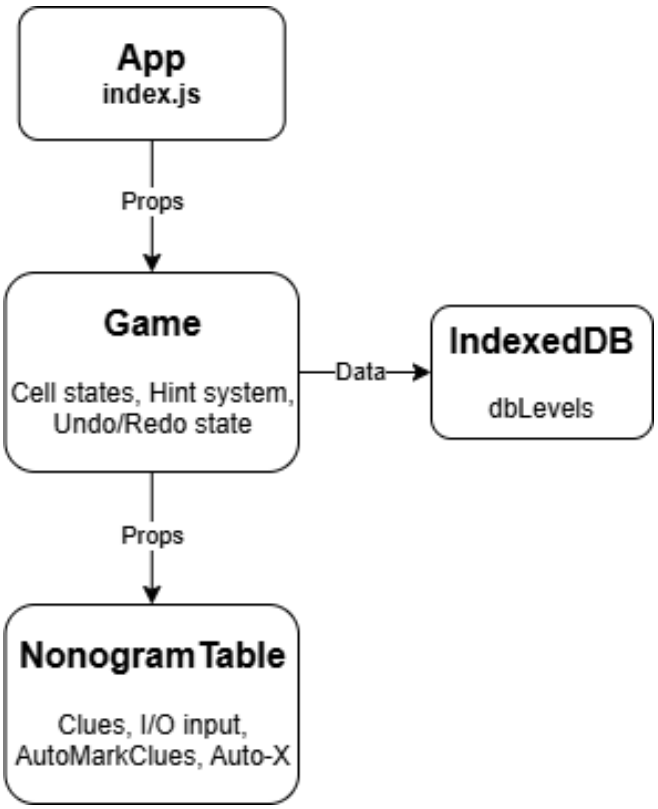
Game.js: Είναι το πιο κεντρικό και κρίσιμο component. Περιλαμβάνει την κύρια λογική του παιχνιδιού: διαχείριση του grid, επιλογή mode, αποθήκευση/φόρτωση, επαλήθευση λύσης, εφαρμογή hints, και αποστολή props σε άλλα components όπως το NonogramTable. Υλοποιεί το game loop, όπως θα αναλυθεί λεπτομερώς στο Κεφάλαιο 3. Η λειτουργία undo/redo πραγματοποιείται μέσω ιστορικού καταστάσεων (historyStack) και χρήσης του useReducer hook.

NonogramTable.js: Είναι το component που προβάλλει το πλέγμα. Κάθε κελί αποδίδεται ως ξεχωριστό div, και χρησιμοποιούνται conditional classNames για την απόδοση της οπτικής κατάστασης (γεμισμένο, κενό, σημειωμένο, σωστό, λάθος). Η απόδοση είναι δυναμική, και το πλέγμα ενημερώνεται μέσω props και callbacks. Για λόγους απόδοσης, χρησιμοποιείται memorization (React.memo) για να περιοριστεί το unnecessary rendering.

Η οργάνωση των αρχείων ακολουθεί την εξής δομή:

1. /components: Περιέχει όλα τα βασικά components της React.
2. /utils: Χρήσιμες συναρτήσεις (π.χ. υπολογισμός clues, deep cloning πλέγματος).
3. /hooks: Custom hooks όπως useUndoRedo.
4. /storage: Επικοινωνία με Dexie και IndexedDB, schema της βάσης.
5. /styles: Αρχεία CSS χωρισμένα ανά component.

Η παραπάνω διάρθρωση διευκολύνει την εργασία με modules, την επαναχρησιμοποίηση κώδικα και τον διαχωρισμό μεταξύ λογικής και παρουσίασης.



Εικόνα 2.1 Διάγραμμα Activity όπου εμφανίζονται τα 2 βασικότερα Components (Game και NonogramTable) και η συσχετισή τους καθώς και η επικοινωνία με την βάση (IndexedDB)

Η συνολική αρχιτεκτονική ακολουθεί την τυπική μορφή μίας σύγχρονης single-page εφαρμογής (SPA), με τα εξής βασικά επίπεδα:

- Επίπεδο διεπαφής (UI Layer): περιλαμβάνει την απεικόνιση πλέγματος, clues και controls (NonogramTable.js, στοιχεία div, κλάσεις CSS).
- Επίπεδο λογικής (Logic layer): περιλαμβάνει την εφαρμοσμένη λειτουργικότητα επίλυσης και διαχείρισης (Game.js, useUndoRedo.js).
- Επίπεδο δεδομένων (data layer): καλύπτει το αποθηκευτικό υπόβαθρο Dexie/IndexedDB και το state management (React hooks, Context).

Ο διαχωρισμός αυτός προσφέρει ευκολία συντήρησης, σαφή ευθύνη σε κάθε υποσύστημα και εξασφαλίζει ότι η εφαρμογή μπορεί να επεκταθεί μελλοντικά, προσθέτοντας νέες λειτουργίες ή εναλλακτικές μορφές αποθήκευσης και rendering χωρίς να διαταραχθεί η υπάρχουσα υποδομή.

2.6 Στρατηγικές Δοκιμών και Εντοπισμού Σφαλμάτων

Η διαδικασία ελέγχου και επιβεβαίωσης της ορθής λειτουργίας της εφαρμογής Nonogram αποτέλεσε κρίσιμο στάδιο της ανάπτυξης, ιδιαίτερα λόγω της φύσης της εφαρμογής ως εργαλείου λογικής

επίλυσης γρίφων. Η ακρίβεια στη συμπεριφορά των clues, στη διαχείριση της κατάστασης του παιχνιδιού και στη διάδραση του χρήστη με το πλέγμα είναι απαραίτητη για να διασφαλιστεί ότι το λογισμικό λειτουργεί αξιόπιστα και προσφέρει συνεπή εμπειρία.

Για την υλοποίηση των στρατηγικών δοκιμών, ακολουθήθηκαν τρεις βασικές κατευθύνσεις: Επιτόπιες δοκιμές (manual testing) κατά τη διάρκεια ανάπτυξης.

- Λογικός διαχωρισμός σε λειτουργικές μονάδες για δυνατότητα μεμονωμένου ελέγχου.
- Ανάλυση edge cases και προσπάθεια αναπαραγωγής σφαλμάτων με ασυνήθιστα δεδομένα.

Κατά την ανάπτυξη των βασικών components της εφαρμογής, χρησιμοποιήθηκαν εργαλεία debugging όπως:

- Η κονσόλα του browser (console.log, console.table) για παρακολούθηση καταστάσεων του πλέγματος, clues και ιστορικού.
- Το εργαλείο React Developer Tools για επιθεώρηση του δέντρου των components, των props και του internal state.
- Το Chrome DevTools Performance tab για μέτρηση χρόνων rendering, ιδίως σε μεγάλα πλέγματα (20x20, 25x25).

Οι δοκιμές πραγματοποιήθηκαν κυρίως σε περιβάλλον Google Chrome (v121+), αλλά επίσης σε Firefox και Edge, για να επιβεβαιωθεί η διαλειτουργικότητα με τις IndexedDB APIs. Εξετάστηκαν τα εξής σενάρια:

- Δημιουργία επιπέδου και επιβεβαίωση της ορθής δημιουργίας clues.
- Εναλλαγή μεταξύ modes και έλεγχος αποθήκευσης της κατάστασης.
- Λειτουργία undo/redo σε γρήγορες αλληλουχίες ενεργειών.
- Αξιοπιστία hints – επιλέγονται σωστά τα λογικά δυνατά κελιά.
- Εμφάνιση σφαλμάτων (invalid input, out of bounds, κενά clues).

Επιπρόσθετα, η δυνατότητα χρήσης της εφαρμογής σε καταστάσεις με περιορισμένη ή καθόλου σύνδεση (offline mode) εξετάστηκε μέσω της λειτουργίας airplane mode και καθαρισμού cache, για επιβεβαίωση της λειτουργίας του IndexedDB και της αποθήκευσης των επιπέδων.

Οι βασικές λειτουργίες που τέθηκαν υπό αυστηρό έλεγχο και περιγράφονται εκτενώς στο Κεφάλαιο 3 είναι:

- Η λειτουργία της undo/redo με ιστορικό ενεργειών και διαχείριση συγκρούσεων.
- Ο μηχανισμός hints και η λογική απόφασης ποιο κελί προτείνεται.
- Η αποθήκευση και ανάκτηση δεδομένων μέσω Dexie.js και IndexedDB.

Η συνεχής χρήση και επαναληπτικός έλεγχος (regression testing) αποτέλεσαν μέσο διασφάλισης ποιότητας, σε συνδυασμό με την modular δομή που διευκόλυνε τον απομονωμένο έλεγχο επιμέρους λειτουργιών. Ο κώδικας δοκιμάστηκε και με εσκεμμένες εισαγωγές λαθών (π.χ. null clues, μη τετράγωνα grids) για αξιολόγηση της ανθεκτικότητας του λογισμικού.

2.7 Προβλήματα και Περιορισμοί

Κατά την ανάπτυξη και τη δοκιμή της εφαρμογής Nonogram εντοπίστηκαν διάφορα τεχνικά και σχεδιαστικά ζητήματα, τα οποία είτε επιλύθηκαν στο πλαίσιο της υλοποίησης, είτε αναγνωρίστηκαν ως περιορισμοί που μπορούν να αντιμετωπιστούν σε μελλοντικές εκδόσεις. Τα προβλήματα αυτά αφορούν κυρίως την επεκτασιμότητα της αρχιτεκτονικής και τις δυνατότητες ενίσχυσης της εμπειρίας χρήστη.

Περιορισμός 1: Έλλειψη έγχρωμων Nonogram

Η εφαρμογή λειτουργεί σε δυαδικό μοντέλο (γεμισμένο / κενό), χωρίς υποστήριξη για έγχρωμα puzzles. Η εισαγωγή έγχρωμης πληροφορίας απαιτεί αλλαγές τόσο στη δομή των κελιών όσο και στο UI (color picker, clues per color) και στην αποθήκευση (IndexedDB schema).

Περιορισμός 2: Έλλειψη multi-user ή cloud αποθήκευσης

Η επιλογή χρήσης Dexie.js προσφέρει πλήρη offline εμπειρία, ωστόσο περιορίζει τον συγχρονισμό και τη μεταφορά επιπέδων μεταξύ συσκευών. Δεν υποστηρίζεται ακόμη σύνδεση με κάποιο backend σύστημα (π.χ. Firebase), ούτε login, ούτε cloud-based sharing.

Περιορισμός 3: Δεν υπάρχει έλεγχος μοναδικότητας λύσης

Κατά τη δημιουργία custom Nonogram, η εφαρμογή δεν ελέγχει αν τα clues οδηγούν σε μοναδική λύση. Αυτό περιορίζει τη χρήση της ως αυστηρό εργαλείο για δημιουργία λογικών puzzles, ιδίως σε εκπαιδευτικά περιβάλλοντα ή περιπτώσεις κοινοποίησης σε άλλους χρήστες.

Τα παραπάνω προβλήματα δεν αναιρούν τη λειτουργικότητα της εφαρμογής, αλλά προσδιορίζουν σημεία προς βελτίωση. Στο Κεφάλαιο 4 παρουσιάζονται προτάσεις επίλυσης, επανασχεδίασης και ενίσχυσης της εμπειρίας χρήστη, βάσει των εμπειριών και των περιορισμών που καταγράφηκαν.

2.8 Επίλογος

Το παρόν κεφάλαιο παρουσίασε αναλυτικά την τεχνική δομή και την αρχιτεκτονική της εφαρμογής Nonogram. Μέσα από τη λεπτομερή παρουσίαση των τεχνολογικών εργαλείων που χρησιμοποιήθηκαν, όπως η ReactJS, η Dexie.js και η IndexedDB, αναδείχθηκε η επιλογή σύγχρονων, αποδοτικών και επεκτάσιμων λύσεων που ανταποκρίνονται πλήρως στις απαιτήσεις μίας διαδραστικής single-page εφαρμογής.

Παράλληλα, περιγράφηκαν οι λειτουργικές καταστάσεις του συστήματος, η αρχιτεκτονική των επιμέρους components και η συνολική δομή του κώδικα, τεκμηριώνοντας την επιλογή modular σχεδίασης. Τονίστηκε η σημασία της σαφούς οργάνωσης των αρχείων και της διαχείρισης κατάστασης, ιδίως σε εφαρμογές που χαρακτηρίζονται από σύνθετες αλληλεπιδράσεις, όπως οι γρίφοι τύπου Nonogram.

Οι στρατηγικές ελέγχου και οι δοκιμές που πραγματοποιήθηκαν συνέβαλαν καθοριστικά στην αξιολόγηση της εφαρμογής, εντοπίζοντας κρίσιμα σημεία προς βελτίωση και διασφαλίζοντας την ομαλή εμπειρία του χρήστη. Τέλος, αναφέρθηκαν προβλήματα και περιορισμοί που προέκυψαν κατά την ανάπτυξη, τα οποία αποτελούν αφετηρία για μελλοντική βελτίωση.

Κεφάλαιο 2ο:

Το επόμενο κεφάλαιο εστιάζει στην ανάλυση των επιμέρους λειτουργιών της εφαρμογής. Συγκεκριμένα, εξετάζεται η λογική υλοποίησης των clues, του μηχανισμού υποδείξεων, της ιστορικότητας (undo/redo) και της επαλήθευσης της λύσης, καθώς και ο τρόπος με τον οποίο αυτά τα επιμέρους στοιχεία συνθέτουν τον πυρήνα της λειτουργικότητας του συστήματος.

Κεφάλαιο 3ο: Τεχνική Ανάλυση και Υλοποίηση Επιμέρους Λειτουργιών

3.1 Εισαγωγή

Στο παρόν κεφάλαιο πραγματοποιείται εις βάθος τεχνική ανάλυση των επιμέρους λειτουργιών της εφαρμογής Nonogram. Έμφαση δίνεται τόσο στη λογική σχεδίαση όσο και στον τρόπο υλοποίησης των βασικών δυνατοτήτων, όπως η παραγωγή αριθμητικών ενδείξεων (clues), η δυνατότητα δημιουργίας νέων επιπέδων μέσω του Edit mode, η χρήση εργαλείων όπως το undo/redo και οι υποδείξεις (hints), καθώς και ο μηχανισμός επαλήθευσης της λύσης.

Η ανάλυση που ακολουθεί τεκμηριώνεται με αναφορές στον πραγματικό κώδικα της εφαρμογής, καθώς και σε παραδείγματα χρήσης και αλληλεπίδρασης μεταξύ των React components που εμπλέκονται. Κάθε λειτουργική ενότητα εξετάζεται ξεχωριστά, παρουσιάζοντας τόσο τον αλγόριθμο που την διέπει όσο και τα τεχνικά μέσα με τα οποία επιτυγχάνεται η επιθυμητή συμπεριφορά εντός του περιβάλλοντος του browser.

Η λειτουργικότητα της εφαρμογής βασίζεται στον αποτελεσματικό χειρισμό της κατάστασης του πλέγματος, της συνεχούς ανανέωσης του UI και της επεξεργασίας εισόδου του χρήστη σε πραγματικό χρόνο. Η υλοποίηση όλων αυτών των στοιχείων υποστηρίζεται από τη χρήση εργαλείων όπως τα React hooks, η τοπική αποθήκευση με IndexedDB, και η αξιοποίηση βοηθητικών συναρτήσεων για τον έλεγχο, τον υπολογισμό και τη σύγκριση δεδομένων.

Κάθε υποενότητα του κεφαλαίου 3 καταγράφει τη λειτουργία, τα δεδομένα εισόδου και εξόδου, τις τεχνικές προκλήσεις που αντιμετωπίστηκαν και τον τρόπο με τον οποίο ενσωματώνονται στο συνολικό αρχιτεκτονικό πλαίσιο της εφαρμογής. Ο αναγνώστης μπορεί να κατανοήσει όχι μόνο το τι κάνει η εφαρμογή, αλλά και πώς υλοποιείται σε επίπεδο λογισμικού, συνδέοντας τη θεωρητική ανάλυση με τον πρακτικό κώδικα.

Το κεφάλαιο αυτό διασαφηνίζει τον μηχανισμό λειτουργίας και τη συνολική δυναμική που διέπει τη σχεδίαση ενός ψηφιακού εργαλείου επίλυσης γρίφων, το οποίο λειτουργεί αυτόνομα στον browser, χωρίς εξωτερικούς εξυπηρετητές και με δυνατότητα offline χρήσης.

3.2 Δομή Πλέγματος και Παραγωγή Ενδείξεων (Grid & Clues)

Η θεμελιώδης δομή δεδομένων της εφαρμογής Nonogram είναι το πλέγμα (grid), το οποίο αποτελεί το επίκεντρο όλων των λειτουργιών: σχεδίαση, επίλυση, υπολογισμός ενδείξεων, έλεγχος εγκυρότητας και υποδείξεις. Η κατανόηση του τρόπου με τον οποίο δομείται και χειρίζεται το πλέγμα είναι ουσιώδης για την κατανόηση του συνολικού συστήματος.

Το πλέγμα αναπαρίσταται εσωτερικά ως ένας δισδιάστατος πίνακας διαστάσεων n x m, όπου n είναι ο αριθμός των γραμμών και m ο αριθμός των στηλών. Κάθε στοιχείο του πίνακα αντιστοιχεί σε ένα συγκεκριμένο κελί του Nonogram και υλοποιείται είτε ως αντικείμενο JavaScript με ιδιότητες που περιγράφουν την κατάστασή του, είτε με μία απλή Boolean τιμή (true/false), που εξαρτάται από την λειτουργική κατάσταση που χρησιμοποιείται (mode).

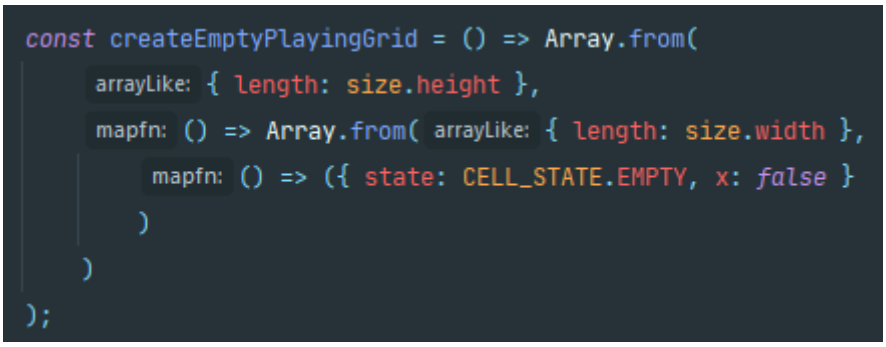
Επομένως η δομή του κάθε κελιού έχει ως εξής:

- **Edit Mode**: κάθε κελί αντιστοιχεί σε μια απλή boolean τιμή true ή false, όπου:
  - **true** σημαίνει ότι το κελί είναι γεμισμένο (μαύρο).
  - **false** σημαίνει ότι το κελί είναι κενό (λευκό).
- **Play Mode**: κάθε κελί αναπαρίσταται ως αντικείμενο με τη δομή:

  - **state**: 0: άδειο, 1: γεμισμένο, 2: λάθος ή επιλεγμένο (ανά context)
  - **x:** boolean αν εχει σημειωθεί από τον χρήστη ως X

Το state δηλώνει την κατάσταση του κελιού από την πλευρά του χρήστη (τι πιστεύει ότι υπάρχει στο κελί), ενώ το x αφορά την προσθήκη σημείωσης, όταν ο χρήστης θεωρεί ότι το κελί δεν πρέπει να γεμιστεί. Στο Play Mode δεν είναι προσβάσιμο το πραγματικό περιεχόμενο (η σωστή απάντηση), παρά μόνο μέσω ελέγχου (π.χ. validate function). Η διαφοροποίηση αυτή στη δομή επιτρέπει στη React να χειρίζεται το ίδιο grid με διαφορετικά μοντέλα δεδομένων ανά mode, εξασφαλίζοντας μεγαλύτερη αποδοτικότητα και ευκολότερη λογική στην εφαρμογή εντολών.

Κατά τη δημιουργία ενός νέου γρίφου ή την απόδοση υπάρχοντος, δημιουργείται ένας νέος πίνακας κελιών μέσω δυναμικής κατασκευής, π.χ. με χρήση:



Η συνάρτηση createEmptyPlayingGrid() επιστρέφει ένα αντικείμενο με τις προεπιλεγμένες τιμές. Ο διαχωρισμός αυτός επιτρέπει την εύκολη επαναχρησιμοποίηση και reset των κελιών, καθώς και την ασφαλή μετατροπή του πλέγματος (π.χ. για deep clone, σύγκριση, αποθήκευση).

Το πλέγμα αποθηκεύεται στον κεντρικό state του component Game.js και μεταδίδεται μέσω props στα θυγατρικά components όπως το NonogramTable. Τα cells με τα οποία παίζει ή σχεδιάζει ο χρήστης (Play ή Edit Modes) αποθηκεύονται στο **cells** state, ενώ υπάρχει και το **originalCells** state όπου αποθηκεύεται ο πίνακας με την λύση. Κάθε κελί αποδίδεται ως ξεχωριστό HTML στοιχείο (div). Ο χρωματισμός, οι ενδείξεις και η διάδραση του χρήστη εξαρτώνται από τις τιμές των ιδιοτήτων του αντικειμένου cell.

Στο UI, γίνεται οπτική διάκριση ανά πέντε κελιά (5x5 blocks), κάτι που επιτυγχάνεται με CSS :nth-child() κανόνες, ώστε να μοιάζει περισσότερο με έντυπο Nonogram. Επίσης, γίνεται χρήση conditional CSS classes για την εμφάνιση του κάθε κελιού ανάλογα με τη λογική του κατάσταση.

Από αυτό το δομημένο πλέγμα προκύπτουν οι αριθμητικές ενδείξεις (clues), οι οποίες είναι η μόνη πληροφορία που έχει ο παίκτης στη διάθεσή του για να επιλύσει τον γρίφο. Οι clues αναπαρίστανται ως πίνακες αριθμών και παράγονται δυναμικά από τη συνάρτηση fillClues(grid).

Για κάθε γραμμή ή στήλη, η συνάρτηση διατρέχει διαδοχικά τα κελιά, μετρώντας συνεχόμενες ακολουθίες γεμισμένων κελιών. Κάθε φορά που το μοτίβο διακόπτεται από κενό (0), ο μετρητής προστίθεται στον πίνακα clues. Εάν δεν εντοπιστεί κανένα γεμισμένο κελί, η ένδειξη είναι [0] ή κενή σειρά. Το αποτέλεσμα για κάθε σειρά ή στήλη είναι ένας πίνακας αριθμών.

Για παράδειγμα:

Είσοδος: [false, true, true, false, true]

Έξοδος: [2, 1]

Τα δεδομένα clues αποθηκεύονται ξεχωριστά ως rowClues και colClues. Επιπλέον, κατά την επίλυση (Play mode), ενημερώνονται αυτόματα ώστε να υποδεικνύουν ποιες γραμμές/στήλες έχουν λυθεί σωστά.

3.3 Δημιουργία Επιπέδου – Edit Mode

Η λειτουργία δημιουργίας επιπέδου (Edit Mode) επιτρέπει στον χρήστη να σχεδιάσει το δικό του Nonogram, καθορίζοντας με ακρίβεια ποια κελιά θα είναι γεμισμένα και ποια κενά. Η διαδικασία αυτή συνδυάζει διάδραση σε πραγματικό χρόνο, ζωντανή ενημέρωση ενδείξεων, μηχανισμό αποθήκευσης και γραφική ανατροφοδότηση. Η υλοποίηση του Edit Mode αποτελεί ένα αυτόνομο αλλά θεμελιώδες υποσύστημα, καθώς παρέχει τα δεδομένα (grid και clues) που χρησιμοποιούνται αργότερα σε λειτουργίες όπως Play ή Preview.

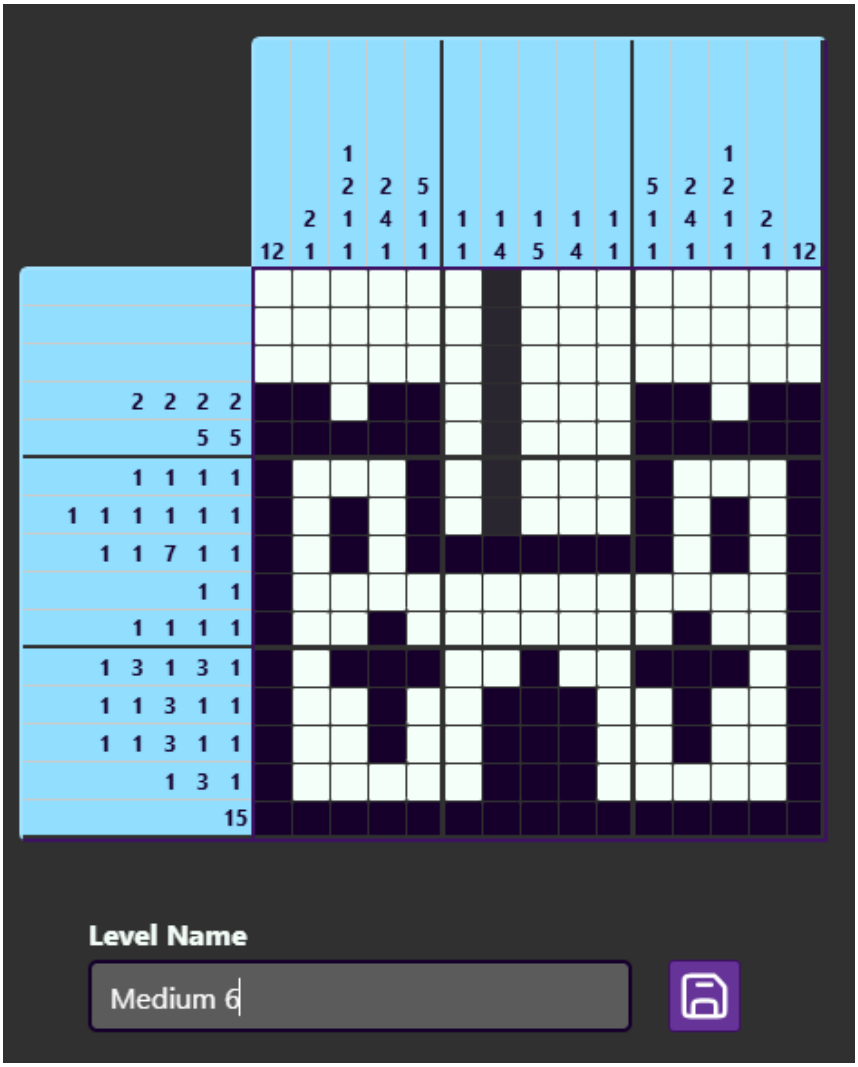
Κατά την είσοδο του χρήστη στο Edit Mode, όπου γίνεται διαλέγοντας το κουμπί Create Nonogram από το κεντρικό μενού, ενεργοποιείται η εσωτερική κατάσταση EDIT στο component Game.js. Ο χρήστης καλείται να καθορίσει τις διαστάσεις του πλέγματος (rows, cols) μέσα από modal παράθυρο. Η δημιουργία του αρχικού πλέγματος γίνεται με τη συνάρτηση createEmptyPlayingGrid(rows, cols), η οποία παράγει έναν πίνακα από boolean τιμές (true/false), που αρχικοποιούνται ως false. Η μεταβλητή cells (React state) ενημερώνεται με το νέο πλέγμα μέσω της setCells.

Η σχεδίαση των κελιών πραγματοποιείται στο component NonogramTable.js, όπου κάθε κελί (cell) αποδίδεται οπτικά ως div. Η διάδραση γίνεται με χρήση event listeners (onMouseDown, onMouseEnter) που καλούν την handleDraw(index) από το Game.js. Η συνάρτηση αυτή εναλλάσσει την τιμή του αντίστοιχου κελιού μεταξύ true (γεμισμένο) και false (κενό). Όταν ο χρήστης σύρει το ποντίκι (drag), εφαρμόζεται το ίδιο μοτίβο σε διαδοχικά κελιά.

Με κάθε αλλαγή στο πλέγμα, γίνεται άμεση επανεκτίμηση των αριθμητικών ενδείξεων (clues). Η συνάρτηση fillClues(grid) καλείται εντός useEffect(() => ..., [cells]), ώστε κάθε αλλαγή να προκαλεί αυτόματο υπολογισμό των rowClues και colClues, τα οποία στη συνέχεια αποδίδονται οπτικά μέσω του CluePanel.js. Έτσι, ο χρήστης μπορεί να βλέπει σε πραγματικό χρόνο την εξέλιξη του γρίφου του και να επαληθεύει την ορθότητα των clues σε σχέση με το σχέδιο που δημιουργεί.

Παράλληλα, παρέχεται δυνατότητα επαναφοράς (reset) μέσω σχετικού κουμπιού στο toolbar, το οποίο καλεί τη setCells(createEmptyPlayingGrid(...)), επαναφέροντας τον πίνακα στην αρχική του κατάσταση. Επίσης, γίνεται έλεγχος για ύπαρξη τουλάχιστον ενός γεμισμένου κελιού προτού επιτραπεί η αποθήκευση.

Για την αποθήκευση νέου επιπέδου υπάρχει μία φόρμα όπου μέσα σε ένα text input ο χρήστης μπορεί να βάλει ένα όνομα για το επίπεδο και πατώντας κουμπί "Save" δίνεται η εντολή. Η αποθήκευση γίνεται με την χρήση Dexie.js, μιας βιβλιοθήκης Wrapper για πιο εύκολη χρήση του IndexedDB API [6]. Πραγματοποιείται η εγγραφή του grid, των clues, και επιπλέον μεταδεδομένων (title, createdAt) στον πίνακα levels της τοπικής IndexedDB βάσης nonogramDB. Το grid μετατρέπεται σε συμβολοσειρά ή serializable μορφή (JSON), ώστε να είναι αποθηκεύσιμο, ενώ τα clues ανακτώνται με τη fillClues κατά την επαναφόρτωση.



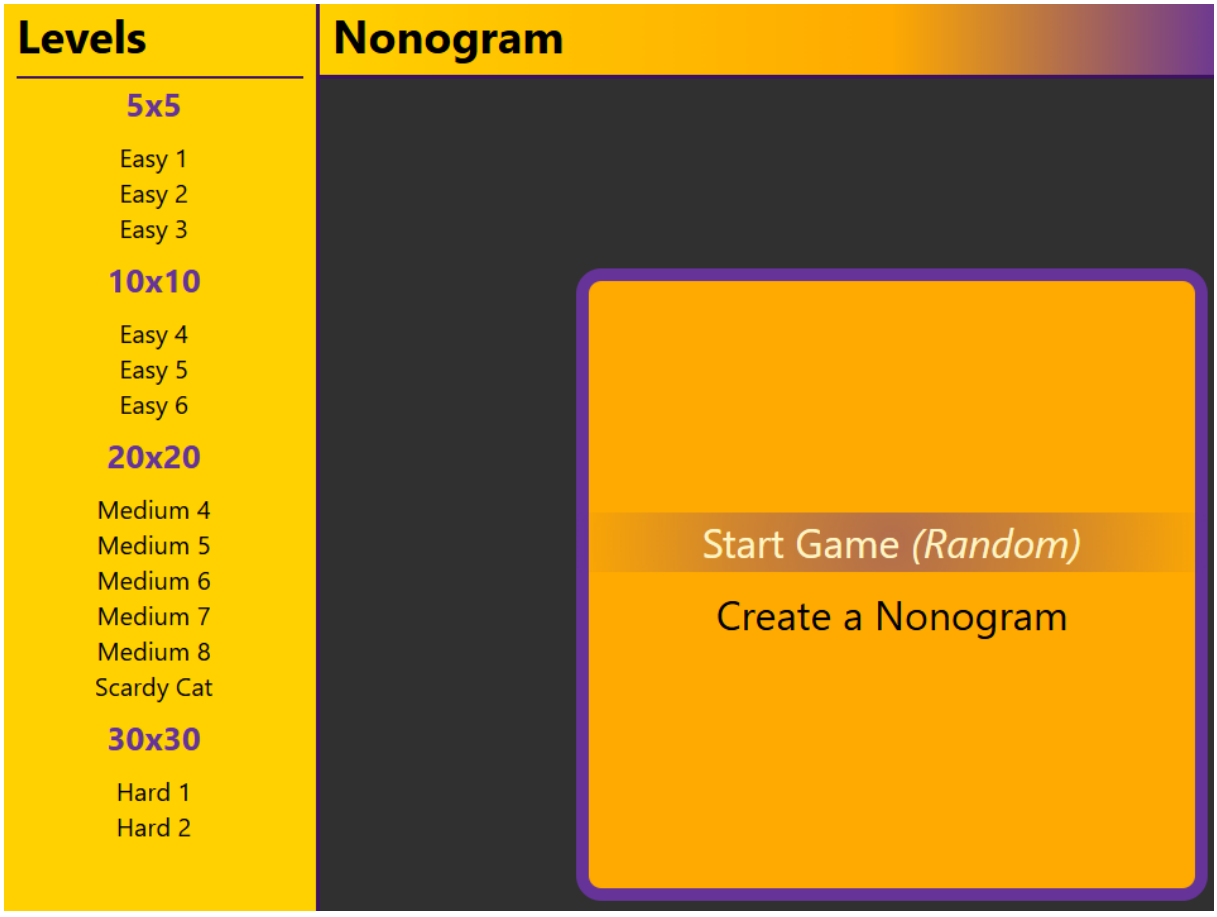
Εικόνα 3.1 Στιγμιότυπο από την εφαρμογή. Φαίνεται το Nonogram σε Edit Mode. Κατω η φόρμα για την αποθήκευση του επιπέδου.

Κατά τη σχεδίαση, γίνεται οπτική ανατροφοδότηση: γεμισμένα κελιά εμφανίζονται με σκούρο φόντο, κενά με λευκό, και το πλέγμα αποδίδεται με grid styling και διαχωριστικές γραμμές ανά πεντάδες. Οι ενδείξεις clues εμφανίζονται στα αριστερά και άνω του πλέγματος και ενημερώνονται άμεσα, χωρίς επαναφόρτωση.

Η λειτουργία Edit Mode επιτρέπει ευελιξία και δημιουργικότητα, προσφέροντας στον χρήστη την αίσθηση ότι "χτίζει" έναν λογικό γρίφο από την αρχή. Η αλληλεπίδραση πλαισιώνεται από απόκριση UI, ζωντανή λογική υποστήριξη και δυνατότητα αποθήκευσης, αποτελώντας βασικό θεμέλιο για την υπόλοιπη λειτουργικότητα της εφαρμογής.

3.4 Επίλυση Επιπέδου – Play Mode

Η λειτουργία Play Mode είναι ο βασικός μηχανισμός της εφαρμογής Nonogram στον οποίο ο χρήστης επιχειρεί την επίλυση ενός γρίφου βάσει των αριθμητικών ενδείξεων (clues) που του παρέχονται. Ο παίκτης δεν γνωρίζει τη λύση εκ των προτέρων και καλείται να τη συμπεράνει χρησιμοποιώντας λογική. Η διεπαφή χρήστη υποστηρίζει πλήρη αλληλεπίδραση με το πλέγμα, σημειώσεις με X, εργαλεία βοήθειας, αναίρεση ενεργειών και δυναμικό έλεγχο προόδου.



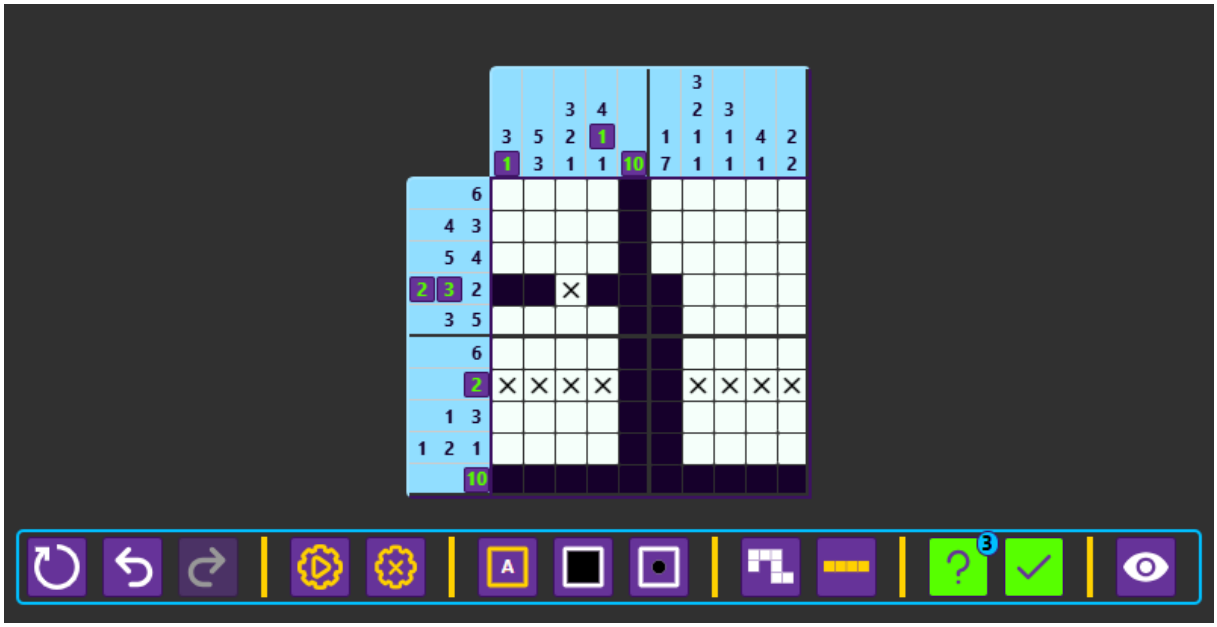
Εικόνα 3.2 Στιγμιότυπο από την εφαρμογή όπου φαίνεται το αρχικό μενού με την επιλογή Start Game (Random) και το πλαϊνό μενού με το σύνολο των υπαρχόντων επιπέδων

Η μετάβαση στο Play Mode εναργοποιείται όταν ο χρήστης διαλέξει ένα επίπεδο από το πλαϊνό μενού ή διαλέξει να λύσει ένα τυχαίο, πατώντας το Start Game (Random) (βλ. Εικόνα 3.2). Το Play Mode ενεργοποιείται μέσω της αλλαγής του state mode σε PLAY στο component Game.js. Κατά την ενεργοποίηση, τα δεδομένα της λύσης (originalCells) φορτώνονται ως boolean πίνακας (true/false), ενώ

το πλέγμα που διαχειρίζεται ο χρήστης (cells) αρχικοποιείται μέσω της συνάρτησης createEmptyPlayingGrid(rows, cols) με αντικείμενα της μορφής:

```
{ state: CELL_STATE.EMPTY, x: false }
```

Επίσης εμφανίζεται ένα Toolbar μενού όπου παρέχει τον χρήστη με διάφορα εργαλεία και βοηθήματα για την επίλυση του γρίφου (βλ. Εικόνα 3.4). Το συγκεκριμένο μενού εμφανίζεται μόνο στο Play Mode και οι λειτουργίες που παρέχει περιγράφονται με περισσότερη λεπτομέρια σε επόμενες ενότητες.



Εικόνα 3.3 Στιγμιότυπο από την εφαρμογή σε Play Mode. Φαίνεται το Toolbar μενού με τις υποστηρικτικές λειτουργίες.

Η σταθερά CELL_STATES ορίζει τρεις καταστάσεις για το πεδίο state:

- **EMPTY (0):** το κελί είναι κενό.
- **FILLED (1):** το κελί είναι γεμισμένο.
- **DOT (2):** το κελί έχει επισημανθεί πιθανό γεμισμένο.

Το πεδίο x είναι boolean και χρησιμοποιείται για να υποδείξει ότι ο χρήστης θεωρεί το κελί κενό, χωρίς να το γεμίσει. Αυτή η επιλογή ενεργοποιείται με δεξί κλικ ή μέσω κατάλληλου εργαλείου σχεδίασης.

Η διάδραση ξεκινά με τη συνάρτηση handleMouseDown(index), η οποία ενεργοποιείται όταν ο χρήστης πατήσει πάνω σε ένα κελί. Καθορίζεται το drawMode ανάλογα με το εργαλείο που έχει ενεργοποιηθεί (π.χ. Fill, X, Dot) και στη συνέχεια καλείται η fillCell(row, col, drawValue) για να ενημερώσει το cells[row][col] σύμφωνα με την επιλογή (βλ. Απόσπασμα Α.1 στο Παράρτημα Α).

Υπάρχει και η δυνατότητα ο χρήστης να γεμίσει πολλαπλά κελιά κρατώντας το κουμπί του ποντικιού. Όταν ο χρήστης κρατά πατημένο το ποντίκι και μετακινείται πάνω σε άλλα κελιά, ενεργοποιείται η handleMouseEnter(row, col) (βλ. Απόσπασμα Α.2 στο Παράρτημα Α). Πιο συγκεκριμένα, η συνάρτηση αυτή ενεργοποιείται όταν ο δείκτης του ποντικιού μετακινηθεί σε άλλα κελιά ενώ είναι πατημένο το κουμπί του ποντικιού (isDrawing = true). Καλεί την ίδια fillCell(...) με την ενέργεια που ξεκίνησε, επιτρέποντας στο χρήστη να γεμίσει ή να σημειώσει συνεχόμενα κελιά κατά τη μετακίνηση. Το

σύστημα χρησιμοποιεί προσωρινή καταγραφή ενεργών κελιών (sessionCells, sessionCellCoords), και σημάνει με συγκεκριμένο χρώμα τα κελιά για τα οποία πρόκειται να αλλάξει το state. Με αυτόν τον τρόπο καταφέρνει να αποτρέψει περιττές εγγραφές στο state.

```
nextState = (current.state + 1) % Object.keys(CELL_STATES).length;
```

Με δεξί κλικ, ανεξάρτητα από την το drawStates, το κελί γεμίζει με X. Παράδειγμα της διάδρασης με

Η λειτουργία αυτή επιτρέπει τη μαζική επεξεργασία κελιών κατά τη μετακίνηση του δείκτη, προσφέροντας στο χρήστη τη δυνατότητα να γεμίσει ή να σημειώσει συνεχόμενα κελιά χωρίς επαναλαμβανόμενα κλικ. Η handleMouseEnter εκτελεί ενέργεια μόνο αν το ποντίκι είναι ενεργό και το εργαλείο έχει επιλεγεί, αποτρέποντας έτσι κατά λάθος αλλαγές κατά την απλή μετακίνηση του δείκτη.

Τα clues παραμένουν ορατά καθ' όλη τη διάρκεια της επίλυσης, αλλά υποστηρίζουν "ενεργή σήμανση" με highlight. Όταν μία γραμμή ή στήλη επιλυθεί σωστά (όλα τα γεμισμένα κελιά είναι σωστά και μόνο αυτά), η συνάρτηση autoMarkClues() αποθηκεύει την πληροφορία ώστε το NonogramTable να επισημάνει τις αντίστοιχες ενδείξεις.

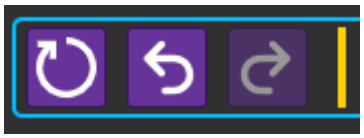
Εκτός από τη βασική δυνατότητα σχεδίασης και επίλυσης, το Play Mode ενσωματώνει και μια σειρά από υποστηρικτικές λειτουργίες που ενισχύουν την εμπειρία χρήστη και διευκολύνουν τη λογική σκέψη, οι οποίες βρίσκονται στο Toolbar μενού (βλ. Εικόνα 3.3):

- **Undo / Redo:** Με χρήση του historyStack, καταγράφεται κάθε αλλαγή στην κατάσταση του πλέγματος. Οι συναρτήσεις handleUndo() και handleRedo() επαναφέρουν τις προηγούμενες ή επόμενες καταστάσεις των cells, προσφέροντας δυνατότητα πειραματισμού χωρίς απώλεια προόδου.
- **Υπόδειξη (Hint):** Μέσω του κουμπιού "Hint", ενεργοποιείται η handleHintClick(), η οποία εντοπίζει κελιά που μπορούν να συμπληρωθούν με βεβαιότητα και το επισημαίνει στο πλέγμα. Η επιλογή βασίζεται στη λογική των clues και της τρέχουσας κατάστασης.
- **Έλεγχος λύσης (Check):** Η συνάρτηση handleCheckSolvedClick() συγκρίνει το πλέγμα του χρήστη (cells) με το originalCells και επισημαίνει τα λάθος κελιά χωρίς να αποκαλύπτει τη λύση.
- **Preview Mode:** Η εφαρμογή επιτρέπει την προσωρινή εναλλαγή σε προεπισκόπηση της σωστής εικόνας του γρίφου μέσω conditional rendering του PreviewModal.js. Η προβολή αυτή δεν επηρεάζει την τρέχουσα επίλυση και χρησιμεύει κυρίως στην επιβεβαίωση του αρχικού σχεδίου από τον δημιουργό.

Αυτές οι λειτουργίες λειτουργούν συμπληρωματικά προς την βασική διαδικασία επίλυσης και συμβάλλουν στη δημιουργία ενός ολοκληρωμένου και υποστηρικτικού περιβάλλοντος λογικής αλληλεπίδρασης.

3.5 Undo / Redo – Ιστορικό Καταστάσεων

Η δυνατότητα αναίρεσης και επαναφοράς ενεργειών (undo / redo) αποτελεί σημαντική προσθήκη στο Play Mode της εφαρμογής Nonogram, προσφέροντας στον χρήστη τη δυνατότητα να πειραματιστεί, να διορθώσει λάθη ή να επιστρέψει σε προηγούμενη κατάσταση χωρίς απώλεια προόδου. Η υλοποίηση του μηχανισμού βασίζεται στο custom React hook useUndoRedo, το οποίο εντοπίζεται στο αρχείο hooks/useUndoRedo.js.



Εικόνα 3.4: Στιγμιότυπο από την εφαρμογή. Reset, Undo και Redo κουμπιά από το Toolbar.

3.5.1 Περιγραφή του useUndeoRedo hook

Το hook useUndoRedo χρησιμοποιεί δύο δομές δεδομένων: undoStack και redoStack, οι οποίες υλοποιούνται ως arrays και διαχειρίζονται με χρήση του useState. Η βασική λογική περιλαμβάνει αποθήκευση snapshots του πλέγματος playingCells κάθε φορά που ο χρήστης πραγματοποιεί μια ενέργεια (μέσω setPlayingCellsWithHistory). Οι προηγούμενες καταστάσεις προστίθενται στο undoStack, ενώ με κάθε undo, η τρέχουσα κατάσταση μεταφέρεται στο redoStack και ανακτάται η αμέσως προηγούμενη.

Η κύρια συνάρτηση του hook για καταγραφή αλλαγών είναι η setPlayingCellsWithHistory(newState, newLevel). Αν το δεύτερο όρισμα είναι true, τότε σηματοδοτείται ένα νέο επίπεδο και γίνεται εκκαθάριση του undoStack. Σε κάθε άλλη περίπτωση, το νέο snapshot του πλέγματος προστίθεται στο ιστορικό, εφόσον υπάρχει ενεργό πλέγμα.

Η αναίρεση (handleUndo) και η επαναφορά (handleRedo) εκτελούνται με βάση το τελευταίο στοιχείο κάθε στοίβας (pop / slice) και ανανεώνουν το playingCells. Παράλληλα, η αντίθετη στοίβα ενημερώνεται με την τρέχουσα κατάσταση ώστε να διατηρείται πλήρες ιστορικό δύο κατευθύνσεων. Το μέγιστο πλήθος ιστορικού ορίζεται στη σταθερά MAX_HISTORY = 50.

Η διαχείριση των αλλαγών στο UI πραγματοποιείται μέσω των συναρτήσεων handleUndoClick() και handleRedoClick() στο Game.js, οι οποίες ενεργοποιούνται με κουμπιά που βρίσκονται στο div className="game__controls". Τα κουμπιά ελέγχουν δυναμικά αν το undoStack.length ή redoStack.length είναι μη μηδενικά ώστε να ενεργοποιούνται ή να απενεργοποιούνται κατάλληλα.

Στο component Game.js, το hook χρησιμοποιείται ως εξής:

```
const [
    playingCells, setPlayingCellsWithHistory,
    undoStack, redoStack,
    handleUndo, handleRedo] = useUndoRedo( initialState: []);
```

Όταν ο χρήστης εκτελέσει κάποια ενέργεια αλλαγής (μέσω fillCell(row, col, drawValue)), καλείται setPlayingCellsWithHistory(updatedCells) και η νέα έκδοση του πλέγματος αντικαθιστά την προηγούμενη. Παράλληλα, η προηγούμενη κατάσταση εισάγεται στη στοίβα undo. Αν ο χρήστης πατήσει το κουμπί Undo, ενεργοποιείται η handleUndo() και στην περίπτωση του Redo, ενεργοποιείται η handleRedo(). Η λογική είναι κυκλική. Κάθε φορά που καλείται handleUndo(), η τρέχουσα κατάσταση αποθηκεύεται στη redoStack, και το τελευταίο στοιχείο της undoStack επαναφέρεται. Αντίστροφα, το handleRedo() επαναφέρει την επόμενη κατάσταση και τοποθετεί την τρέχουσα στο undo. Έτσι διατηρείται πλήρες ιστορικό δύο κατευθύνσεων. Και οι δύο συναρτήσεις παρουσιάζονται αναλυτικά στο Παράρτημα Α.3.

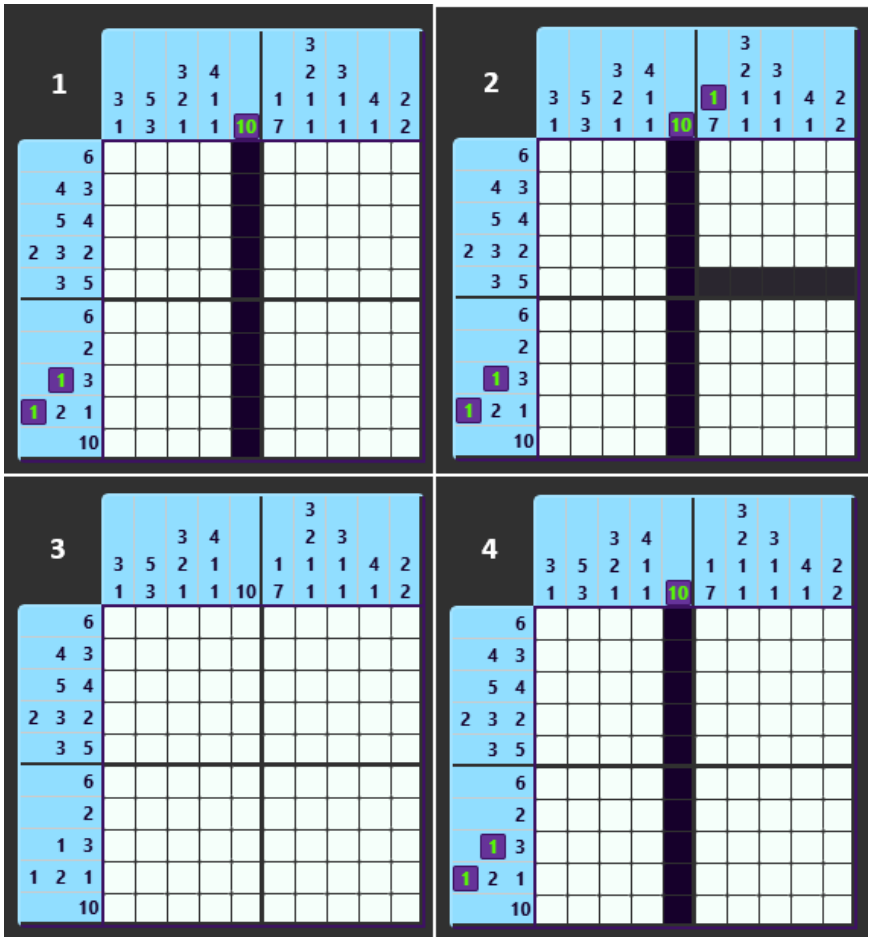
Εσωτερικά, κάθε snapshot αποτελεί deep copy του πίνακα cells, ώστε να διασφαλίζεται ότι δεν υπάρχει αναφορά στα ίδια αντικείμενα. Υπάρχει επίσης και τρόπος για περιορισμό του μεγέθους του undoStack με την μεταβλητή MAX_HISTORY μέσα στο hook.

Για περαιτέρω τεχνικές λεπτομέρειες του hook useUndoRedo και την αρχιτεκτονική των στοίβων undoStack και redoStack, βλ. ενότητα 3.10.7.

3.5.2 Παράδειγμα χρήσης

Παρακάτω παρατίθεται ένα απλό παράδειγμα χρήσης:

1. **Ο χρήστης γεμίζει τα κελιά από [1][5] μέχρι [10][5]** → νέα κατάσταση καταγράφεται.
2. **Γεμίζει τα κελιά [5][6] μέχρι [5][10]**→ νέο snapshot αποθηκεύεται στο undoStack.
3. **Πατά δύο φορές "Undo"** → τα προηγούμενα κελιά εξαφανίζονται.
4. **Πατά "Redo"** → τα κελιά από το 1ο βήμα επαναεμφανίζονται.



Εικόνα 3.5 Στιγμιότυπο από την εφαρμογή με το παράδειγμα χρήσης Undo/Redo.

Το σύστημα undo/redo είναι πλήρως ανεξάρτητο από τον υπόλοιπο μηχανισμό clues ή validation. Δεν γίνεται έλεγχος για ορθότητα κατά την αναίρεση, ούτε γίνεται reset των clues. Το μόνο που αλλάζει είναι η κατάσταση του cells, το οποίο με τη σειρά του ενημερώνει το UI και τις συναρτήσεις παρακολούθησης (π.χ. autoMarkClues).

Η προσέγγιση με custom hook επιτρέπει την επεκτασιμότητα: η λογική του ιστορικού μπορεί να επαναχρησιμοποιηθεί και σε άλλες πτυχές της εφαρμογής, ενώ παρέχει ευελιξία για επέκταση (π.χ. καθολικό undo, per-mode ιστορικό, αποθήκευση στο IndexedDB).

3.6 Μηχανισμός Υποδείξεων (Hint System)

Ο μηχανισμός υποδείξεων (Hint System) της εφαρμογής έχει ως στόχο να παρέχει ουσιαστική βοήθεια στον παίκτη κατά την επίλυση ενός Nonogram, χωρίς όμως να αφαιρεί από τη λογική πρόκληση του παιχνιδιού. Η υπόδειξη δεν λειτουργεί ως αυτόματη λύση του γρίφου, αλλά ως εργαλείο καθοδήγησης σε περιπτώσεις όπου ο παίκτης έχει κολλήσει ή δυσκολεύεται να προχωρήσει.

Σε αντίθεση με ένα απλό “random hint”, ο μηχανισμός έχει σχεδιαστεί να δίνει προτεραιότητα σε περιοχές του πλέγματος όπου ο χρήστης έχει πρόσφατα αλληλεπιδράσει. Αυτό προσφέρει μεγαλύτερη συνέπεια και βοηθά τον παίκτη να παραμείνει συγκεντρωμένος στο σημείο που ήδη προσπαθεί να λύσει. Οι υποδείξεις εφαρμόζονται απευθείας στο πλέγμα, ενώ παράλληλα επισημαίνονται οπτικά τόσο τα εμπλεκόμενα κελιά όσο και οι σχετικές ενδείξεις (clues).

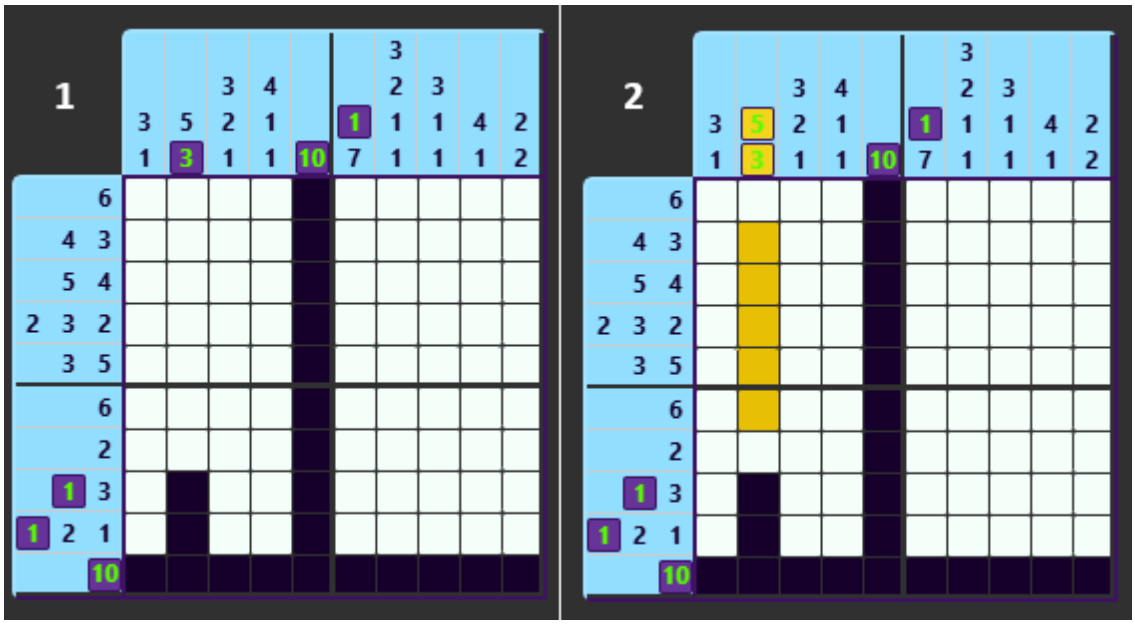
Η προσέγγιση αυτή βασίζεται στη λογική ότι οι υποδείξεις πρέπει να υποστηρίζουν τη σκέψη του παίκτη — όχι να την αντικαθιστούν.

3.6.1 Λειτουργική Περιγραφή

Ο μηχανισμός υπόδειξης ενεργοποιείται όταν ο χρήστης πατήσει το κουμπί “Hint” στο Play Mode. Αν υπάρχουν διαθέσιμες υποδείξεις (hintsCount > 0), τότε το σύστημα εντοπίζει ένα συγκεκριμένο τμήμα (segment) του πλέγματος που δεν έχει ακόμα συμπληρωθεί σωστά και προχωρά στη λήψη μίας υπόδειξης, η οποία οδηγεί σε αυτόματη συμπλήρωση του τμήματος.

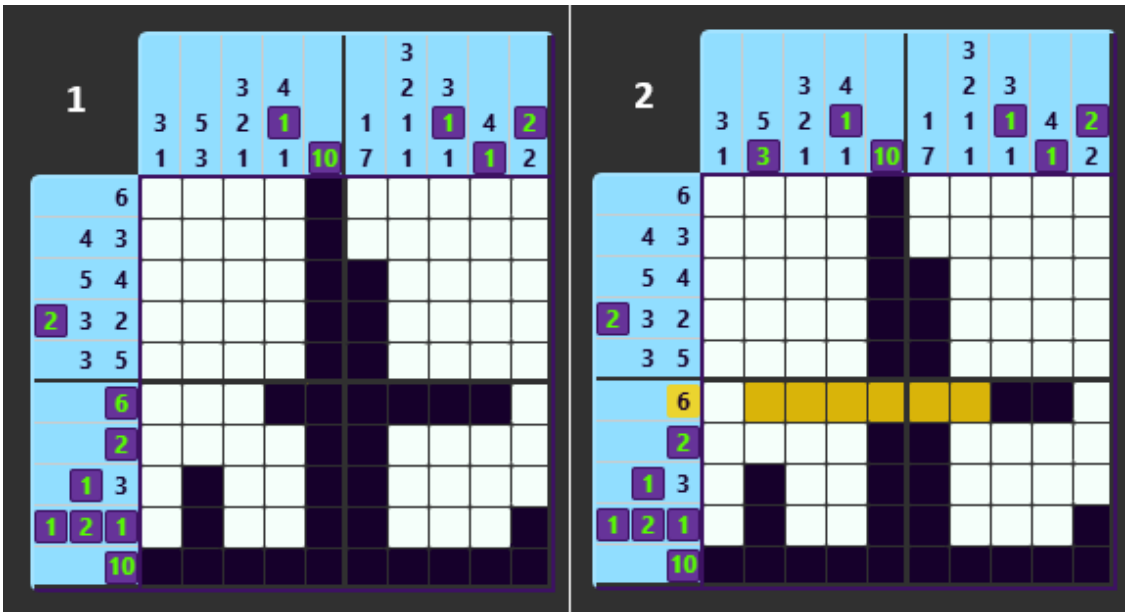
Το σύστημα προτεραιοποιεί περιοχές που σχετίζονται με την τελευταία ενέργεια του παίκτη (lastFilledCell), π.χ. την πιο πρόσφατη σειρά ή στήλη στην οποία συμπλήρωσε κελί. Έτσι, η υπόδειξη σχετίζεται άμεσα με το πλαίσιο επίλυσης που ήδη έχει ξεκινήσει ο παίκτης, διατηρώντας τη ροή σκέψης.

Παράδειγμα 1: Αν ο παίκτης έχει γεμίσει πρόσφατα ένα κελί στην στήλη 2, και αυτή η στήλη περιέχει ακόμα ένα τμήμα (π.χ. 5 συνεχόμενων κελιών) που είτε δεν έχουν συμπληρωθεί καθόλου, είτε έχουν συμπληρωθεί λανθασμένα, το σύστημα θα επιλέξει κατά προτεραιότητα αυτό το segment και θα το γεμίσει, τονίζοντάς το οπτικά (Εικόνα 3.6)



Εικόνα 3.6 Στιγμιότυπο από την εφαρμογή. Ο χρήστης μόλις συμπλήρωσε τα 3 τελευταία κελιά της 2ης στήλης (1). Πατώτας το Hint κουμπί το σύστημα θα γεμίση την υπόλοιπη στήλη με τα 5 συνεχόμενα κελιά (2).

Παράδειγμα 2: Αν ο παίκτης έχει προσπαθήσει να γεμίσει μια γραμμή, αλλά έχει κάνει λάθος — δηλαδή έχει συμπληρώσει κελιά σε λάθος θέση σε σχέση με τη λύση — τότε το σύστημα hint αγνοεί τα ήδη λανθασμένα γεμίσματα και εστιάζει στα σωστά segments που δεν έχουν καλυφθεί. Για παράδειγμα, έστω ότι η γραμμή 6 περιλαμβάνει ένα segment από 6 συνεχόμενα σωστά κελιά, εκ των οποίων ο παίκτης έχει γεμίσει κελιά σε θέσεις που δεν αντιστοιχούν στο σωστό segment. Το σύστημα θα εξακολουθήσει να προσφέρει υπόδειξη για το σωστό segment, παρακάμπτοντας τα λάθη του χρήστη (Εικόνα 3.7). Αυτό βοηθάει σε καταστάσεις όπου ο παίκτης έχει "μπερδευτεί", προσφέροντας μία ξεκάθαρη διόρθωση χωρίς να ακυρώνει τη δική του λογική προσπάθεια.



Εικόνα 3.7 Στιγμιότυπο από την εφαρμογή. Ο χρήστης μόλις συμπλήρωσε τα 6 κελιά στην 6η γραμμή που όμως είναι σε λάθος θέση (1). Το σύστημα θα συμπληρώσει τα 6 αυτά κελιά, με προτεραιότητα, χωρίς να πειράξει τις αλλαγές του χρήστη (2)

Παράδειγμα 3: Όταν δεν έχει υπάρξει πρόσφατο γέμισμα κελιού (lastFilledCell είναι null) από τον παίκτη (δηλαδή δεν έχει καταγραφεί κάποια χρήση του εργαλείου Fill ή X), το σύστημα ακολουθεί εναλλακτική στρατηγική: εξετάζει όλα τα διαθέσιμα μη γεμισμένα τμήματα (Unfilled Segments) στο grid και επιλέγει εκείνο με το μεγαλύτερο μήκος.

Αν υπάρχουν πολλά segments ίδιας έκτασης, τότε το σύστημα επιλέγει τυχαία ένα από αυτά. Έτσι διασφαλίζεται ότι η υπόδειξη παραμένει ουσιαστική και δεν προκύπτει από σταθερή ή στατιστική προκατάληψη. Αυτό το fallback σενάριο εξασφαλίζει πως κάθε υπόδειξη θα έχει νόημα.

Μόλις επιλεγεί ένα κατάλληλο segment:

- Το πλέγμα ενημερώνεται αυτόματα και τα αντίστοιχα κελιά γεμίζουν.
- Τα συγκεκριμένα κελιά επισημαίνονται προσωρινά με διακριτικό χρώμα ή animation, μέσω CSS class (π.χ. nonogramtable__cell--hinted).
- Οι αντίστοιχες ενδείξεις clues (σε γραμμή ή στήλη) εμφανίζονται και αυτές με χρωματική επισήμανση (π.χ. nonogramtable__clues__clue--hinted).
- Ο αριθμός διαθέσιμων υποδείξεων μειώνεται κατά 1 και εμφανίζεται ενημερωμένος πάνω στο κουμπί “Hint”.

Η επισήμανση των υποδεικνυόμενων κελιών δεν παραμένει μόνιμη. Απενεργοποιείται αμέσως μόλις ο χρήστης προβεί σε νέα ενέργεια, π.χ. πατώντας ή γεμίζοντας άλλο κελί, ώστε να αποφευχθεί σύγχυση. Με αυτόν τον τρόπο, η υπόδειξη λειτουργεί σαν στιγμιαία βοήθεια και όχι ως διαρκής υποκατάσταση της λογικής του παίκτη.

3.6.2 Ροή εκτέλεσης

Η βασική λειτουργία του συστήματος υποδείξεων βρίσκεται στη συνάρτηση handleHintClick (βλ. Απόσπασμα Α.4 και Α.5 στο Παράρτημα Α), η οποία ενεργοποιείται κάθε φορά που ο χρήστης πατά το κουμπί "Hint" κατά την επίλυση ενός γρίφου. Η συνάρτηση ακολουθεί μια σειρά βημάτων, ώστε να παρέχει ουσιαστική βοήθεια με λογική συνέπεια και προτεραιότητα.

Η εκτέλεση περιλαμβάνει τα εξής στάδια:

1. **Έλεγχος διαθεσιμότητας:**

   Πριν εκτελεστεί οποιαδήποτε λογική, γίνεται έλεγχος αν υπάρχει τουλάχιστον μία διαθέσιμη υπόδειξη:

   if (hintsCount > 0)

   Αν δεν υπάρχουν διαθέσιμες υποδείξεις (hintsCount === 0), η συνάρτηση τερματίζει άμεσα χωρίς να κάνει αλλαγές στο grid.

2. **Εύρεση μη ολοκληρωμένων segments:**

   Καλείται η βοηθητική getUnfilledSegments(originalCells, playingCells), η οποία επιστρέφει ένα array από αντικείμενα της μορφής:

{ type: 'row' | 'col', index: number, start: number, end: number }

Τα segments αυτά αντιστοιχούν σε διαστήματα συνεχόμενων κελιών που θα έπρεπε να έχουν συμπληρωθεί, αλλά **δεν έχουν** ακόμα στο playing grid.

Η συνάρτηση εξετάζει σειρές και στήλες ξεχωριστά, εντοπίζοντας περιοχές που:

- είναι γεμισμένες στη λύση (originalCells),
- αλλά είναι **κενές** ή μερικώς λανθασμένες στο grid του παίκτη (playingCells).

3. **Προτεραιοποίηση τελευταίας ενέργειας:**

   Αν υπάρχει lastFilledCell, δηλαδή ο χρήστης έχει αλληλεπιδράσει πρόσφατα με κάποιο κελί, η συνάρτηση:

   - φιλτράρει τα segments που ανήκουν στη σειρά ή στήλη της τελευταίας ενέργειας,
   - επιλέγει το μεγαλύτερο από αυτά (μεγαλύτερη διαφορά end - start).

   Αυτό ενισχύει τη λογική συνοχή, αφού η υπόδειξη σχετίζεται με τη σκέψη που ήδη ξεκίνησε ο χρήστης.

4. **Εναλλακτική επιλογή (fallback):**

   Αν δεν βρεθεί κατάλληλο segment με βάση το lastFilledCell (ή είναι null), τότε:

   - γίνεται αναζήτηση του μεγαλύτερου σε μήκος segment,
   - αν υπάρχουν πολλά ίδιας έκτασης, γίνεται τυχαία επιλογή ενός από αυτά.

   Αυτό εξασφαλίζει ότι πάντα θα υπάρξει μια λογική και χρήσιμη υπόδειξη, ακόμη και χωρίς πρόσφατη δράση από τον χρήστη.

5. **Εφαρμογή υπόδειξης στο πλέγμα:**

   Αφού προσδιοριστεί το επιλεγμένο segment:

   - Δημιουργείται ένα αντίγραφο του playing grid (updated) για λόγους immutability.
   - Τα αντίστοιχα κελιά (row-col) γεμίζονται με: **{ state: FILLED, x: false }**
   - Τα ids των κελιών προστίθενται στο newHintCells, ένα Set με format "row-col".

6. **Ενημέρωση κατάστασης (state):**

   Η υπόδειξη εφαρμόζεται μέσω των παρακάτω state ενημερώσεων:

   - Ενημέρωση πλέγματος: **setPlayingCellsWithHistory()**
   - Προσωρινή σήμανση κελιών: **setHintCells(new Set(newHintCells))**
   - Μείωση υποδείξεων: **setHintsCount(count => count - 1)**
   - Σήμανση των clues: **setHintedClue({ type: ..., index: ... })**

Η υπόδειξη που προσφέρεται μέσω του μηχανισμού Hint δεν λειτουργεί απλώς ως συμβουλή ή ένδειξη, αλλά εφαρμόζει απευθείας την ορθή λύση σε συγκεκριμένα κελιά. Αυτό αποτελεί συνειδητή σχεδιαστική επιλογή: στόχος είναι να βοηθηθεί ο παίκτης σε σημεία που έχει “κολλήσει”, προσφέροντάς του μια άμεση ώθηση χωρίς να διακόπτει τη συνολική ροή σκέψης. Ο περιορισμένος

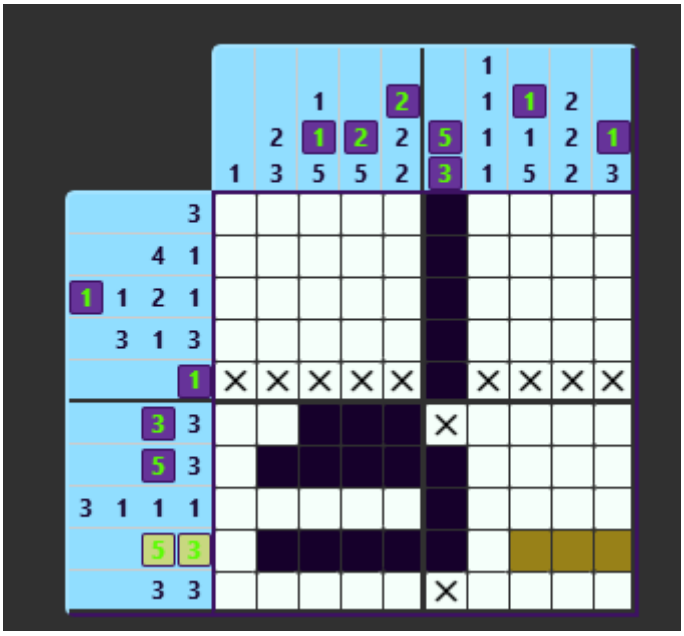
αριθμός διαθέσιμων υποδείξεων (hintsCount) εξασφαλίζει ότι η χρήση του μηχανισμού παραμένει ισορροπημένη και δεν καθιστά περιττή τη λογική προσπάθεια του χρήστη.

3.6.3 Ενημέρωση Grid και Καταστάσεων (States)

Αφού εντοπιστεί το κατάλληλο segment, η συνάρτηση handleHintClick προχωρά στην άμεση εφαρμογή της υπόδειξης στο πλέγμα. Αυτό γίνεται με τη δημιουργία ενός νέου grid (updated), το οποίο προκύπτει από αντιγραφή του υπάρχοντος playingCells, ώστε να διατηρηθεί η ακεραιότητα του αρχικού πίνακα.

Στο νέο αυτό grid, τα κελιά που αντιστοιχούν στο επιλεγμένο segment ενημερώνονται ώστε να εμφανίζονται ως γεμισμένα (**state: FILLED**) και χωρίς X (**x: false**). Η επιλογή αυτή γίνεται είτε σε γραμμή είτε σε στήλη, ανάλογα με τον τύπο του segment (type: "row" ή "col") και τη θέση του (index, start, end).

Παράλληλα, δημιουργείται ένα νέο σύνολο hintCells, το οποίο περιέχει τα row-col identifiers ("3-5", "6-2", κ.λπ.) των κελιών που αφορούν την υπόδειξη. Το σύνολο αυτό αξιοποιείται από το NonogramTable για την οπτική επισήμανση αυτών των κελιών (με CSS class .nonogramtable__cell--hinted), κάνοντας τη βοήθεια ευδιάκριτη στον χρήστη (βλ. Εικόνα 3.8).



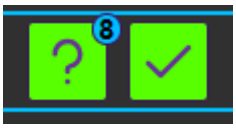
Εικόνα 3.8 Στιγμιότυπο από την εφαρμογή. Φαίνεται η επισήμανση τόσο των κελιών όσο και των clues από τον μηχανισμό υποδείξεων (Hint)

Επιπλέον, η μεταβλητή hintedClue ενημερώνεται με πληροφορίες για την πλευρά (σειρά ή στήλη) στην οποία εφαρμόστηκε η υπόδειξη:

{ type: "row", index: 3 } ή { type: "col", index: 7 }

Με τον τρόπο αυτό, εκτός από το grid, επισημαίνονται και οι αντίστοιχες αριθμητικές ενδείξεις (clues) στην αντίστοιχη πλευρά του πλέγματος — π.χ. η στήλη 7 αν το hint αφορούσε σε αυτήν.

Τέλος, ο αριθμός διαθέσιμων υποδείξεων (hintsCount) μειώνεται κατά 1 και η νέα του τιμή εμφανίζεται άμεσα σαν ετικέτα, πάνω δεξιά, στο κουμπί Hint.



Εικόνα 3.9 Το κουμπί hint (αριστερά). Φαίνεται η ετικένα με τον αριθμό των υπολειπόμενων υποδείξεων.

Η υλοποίηση του συστήματος υπόδειξης επιτυγχάνει να προσφέρει ουσιαστική βοήθεια στον παίκτη χωρίς να ακυρώνει τη λογική πρόκληση του παιχνιδιού. Μέσα από την αξιοποίηση του ιστορικού ενεργειών, την ανάλυση της τρέχουσας προόδου και την οπτική επισήμανση στο περιβάλλον του grid, το σύστημα ενσωματώνεται φυσικά στην εμπειρία επίλυσης. Το αποτέλεσμα είναι ένας μηχανισμός που λειτουργεί υποστηρικτικά, ειδικά σε δύσκολα σημεία, διατηρώντας παράλληλα τον έλεγχο στα χέρια του παίκτη.

3.7 Έλεγχος Λύσης

Ο έλεγχος της ορθότητας της λύσης αποτελεί βασική λειτουργία του Play Mode στην εφαρμογή Nonogram. Η λειτουργία αυτή παρέχει στον χρήστη τη δυνατότητα να ελέγξει αν έχει συμπληρώσει σωστά το πλέγμα βάσει των clues που του έχουν δοθεί, χωρίς όμως να αποκαλύπτεται η τελική εικόνα ή να παραβιάζεται η λογική πρόκληση του παιχνιδιού.

3.7.1 Περιγραφή του ελέγχου λύσης

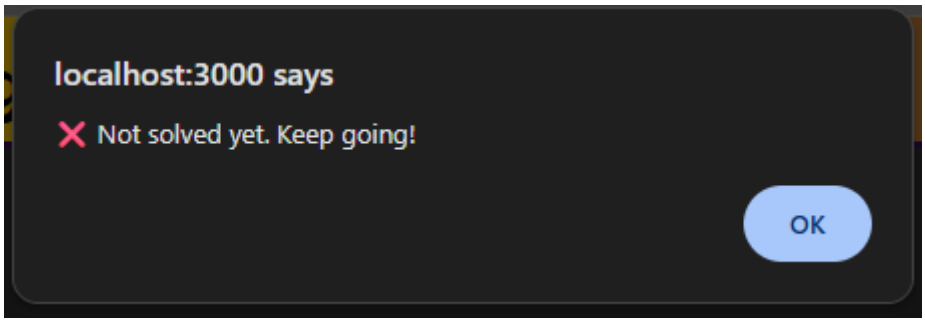
Η ενεργοποίηση του ελέγχου γίνεται μέσω της συνάρτησης handleCheckSolvedClick() η οποία βρίσκεται ορισμένη μέσα στο αρχείο Game.js. Ουσιαστικά, πρόκειται για μία λογική σύγκρισης μεταξύ δύο πινάκων: του πίνακα της λύσης (originalCells) και του πίνακα της χρήστης (playingCells). Αν και τα δύο grids είναι ισοδύναμα ως προς τη λογική διάταξη γεμισμένων κελιών, τότε η εφαρμογή αναγνωρίζει την επίλυση ως επιτυχημένη.

Η διαδικασία ξεκινά με χρήση του μεθόδου every() σε κάθε γραμμή του πίνακα. Η συνάρτηση εξετάζει αν όλα τα στοιχεία (κελιά) μιας γραμμής είναι ίδια μεταξύ των δύο πινάκων, με βασικό κριτήριο αν το state του κάθε κελιού στο playingCells είναι FILLED μόνο όταν το αντίστοιχο κελί στο originalCells είναι επίσης FILLED, και αντίστροφα.

Εφόσον η σύγκριση επιστρέψει true, η εφαρμογή εμφανίζει ένα μήνυμα επιτυχίας ("Puzzle Solved!") με χρήση της μεθόδου alert(...). Στη συνέχεια ενημερώνεται η μεταβλητή isSolved σε true, η οποία επιδρά σε ολόκληρη τη διεπαφή:

- Απενεργοποιεί τα κουμπιά αλλαγής κατάστασης κελιών.
- Απενεργοποιεί τη δυνατότητα hints, undo/redo και reset.
- Καθαρίζει τα highlights από προηγούμενα hints μέσω setHintCells(new Set()) και setHintedClue(null).

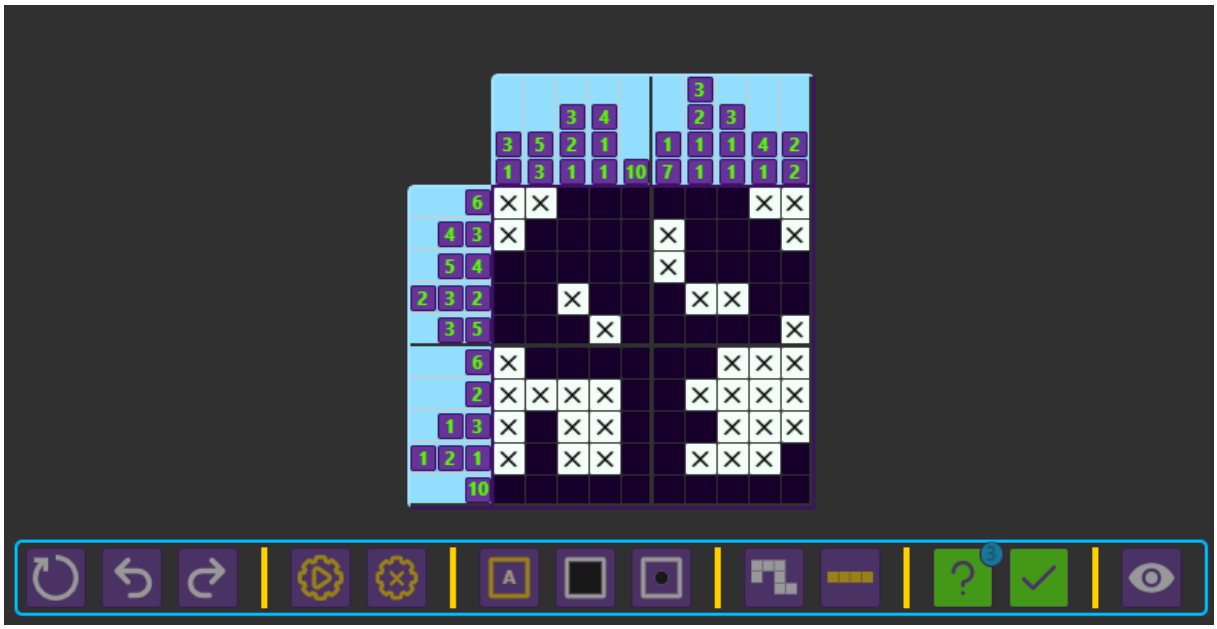
Σε αντίθετη περίπτωση, εμφανίζεται μήνυμα αποτυχίας ("Not solved yet. Keep going!") (βλ. Εικόνα 3.10) και δεν πραγματοποιείται καμία αλλαγή στο state ή στο UI. Ο χρήστης μπορεί να συνεχίσει την επίλυση, να χρησιμοποιήσει τις λειτουργίες hints ή να κάνει undo ώστε να επανέλθει σε προηγούμενες καταστάσεις.



Εικόνα 3.10 Στιγμιότυπο από την εφαρμογή. Alert παράθυρο όταν δεν έχει βρεθεί ακόμη η λύση

Η λειτουργία ελέγχου δεν βασίζεται στα clues (rowClues, colClues) ή σε σήμανση done: true. Είναι αυστηρά λογική, βασισμένη μόνο στην πραγματική θέση των FILLED κελιών και συγκρίνει με απόλυτη ισότητα. Η ύπαρξη DOT ή X δεν επηρεάζει την κρίση αν το κελί θεωρείται σωστά συμπληρωμένο — μόνο το πεδίο state εξετάζεται.

Επιπλέον, η τιμή του isSolved διατηρείται στο state και μπορεί να χρησιμοποιηθεί ως φίλτρο για να εμποδιστούν μελλοντικές ενέργειες, να εμφανιστούν συγχαρητήρια, ή να καταγραφεί πρόοδος. Η χρήση του συνδυάζεται με conditional rendering και με απενεργοποίηση των πλήκτρων του Toolbar (βλ. Εικόνα 3.11). Η handleCheckSolvedClick() συνάρτηση παρουσιάζεται αναλυτικά στο Παράρτημα Α.6.



Εικόνα 3.11 Στιγμιότυπο από την εφαρμογή. Φαίνονταιι τα απενεργοποιημένα πλήκτρα στο Toolbar αφού έχει πατήσει ο χρήστης τον έλεγχο λύσης και έχει λυθεί.

Ο συγκεκριμένος μηχανισμός validation χαρακτηρίζεται από σταθερότητα, επαναληψιμότητα και προβλεψιμότητα. Δεν βασίζεται σε εξωτερικούς παράγοντες, παρά μόνο στο πώς έχει συμπληρώσει ο χρήστης το grid. Παράλληλα, δεν διαρρέει καμία πληροφορία για τη λύση: το μόνο που παρέχει είναι δυαδική απάντηση (σωστό ή όχι), διατηρώντας έτσι το επίπεδο πρόκλησης.

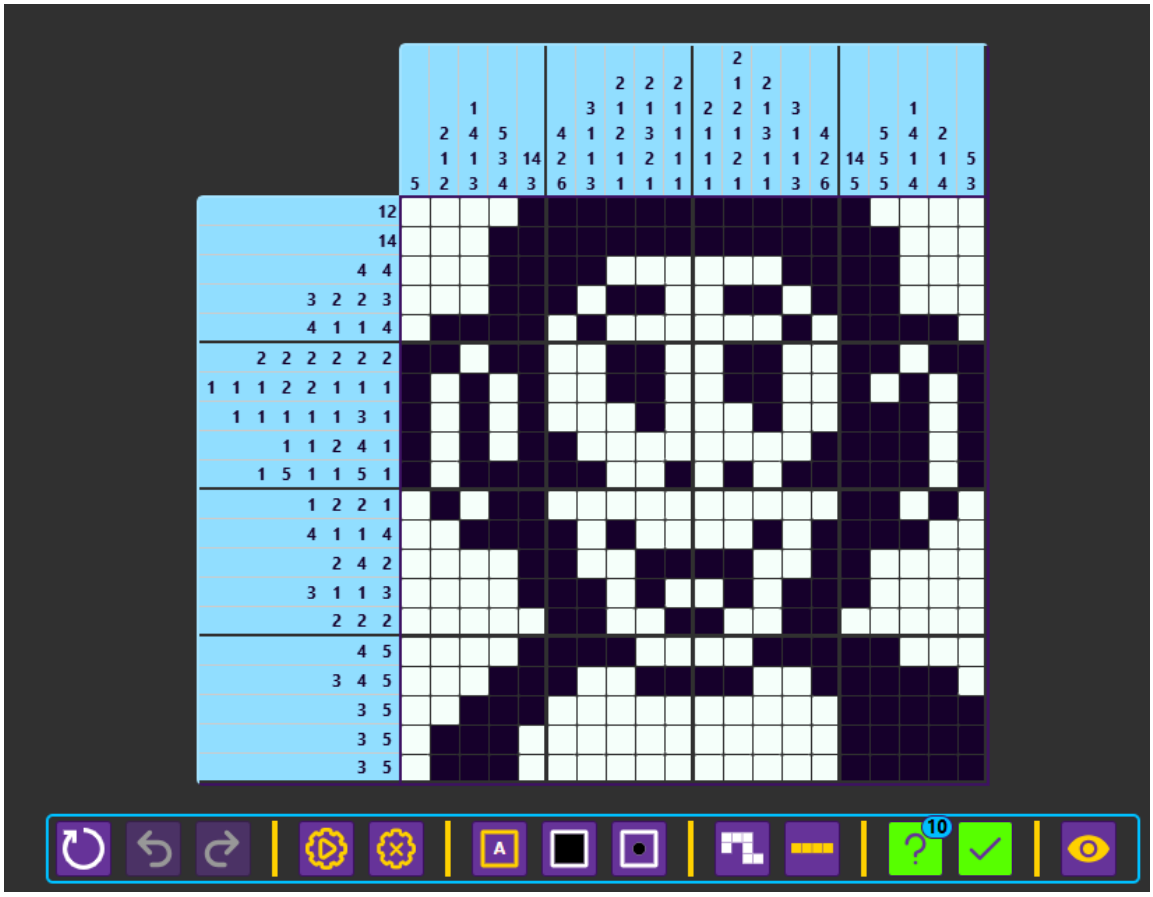
3.8 Προεπισκόπηση – Preview Mode

Η λειτουργία προεπισκόπησης (Preview Mode) αποτελεί συμπληρωματικό εργαλείο στην εφαρμογή Nonogram και επιτρέπει στον χρήστη να δει την πλήρη εικόνα του γρίφου όπως αυτή προκύπτει από το grid των γεμισμένων κελιών. Η λειτουργία αυτή είναι ιδιαίτερα χρήσιμη κατά τη φάση σχεδίασης (Edit Mode), προκειμένου να επιβεβαιώσει ο δημιουργός την ορθότητα και την αισθητική του τελικού αποτελέσματος.

3.8.1 Περιγραφή του Preview Mode

Η μετάβαση στην προεπισκόπηση επιτυγχάνεται μέσω του κουμπιού Preview που ενεργοποιεί τη συνάρτηση handlePreviewClick() στο αρχείο Game.js. Η συνάρτηση αυτή εναλλάσσει την τιμή της μεταβλητής mode μεταξύ modes.PLAY και modes.PREVIEW, η οποία ελέγχει ποιο grid θα αποδοθεί στη διεπαφή χρήστη. Η αλλαγή δεν επηρεάζει τα αποθηκευμένα δεδομένα του grid και διατηρεί την πρόοδο του χρήστη ως έχει.

Η υλοποίηση του οπτικού διαχωρισμού μεταξύ Play και Preview πραγματοποιείται μέσω της βιβλιοθήκης ReactCardFlip, η οποία εφαρμόζει animation δύο όψεων (front/back) με χρήση conditional rendering. Το grid αποδίδεται μέσω του component NonogramTable, είτε με το playingCells (Play Mode), είτε με το originalCells (Preview Mode). Οι δύο όψεις περιλαμβάνονται στο στοιχείο ReactCardFlip και εναλλάσσονται ανάλογα με την τιμή του mode.



Εικόνα 3.12 Στιγμιότυπο από την εφαρμογή. Preview Mode με το Preview κουμπι ενεργοποιημένο.

Στο Preview Mode:

- Δεν επιτρέπεται καμία μορφή αλληλεπίδρασης.
- Το grid αποδίδεται με βάση τα δεδομένα της σωστής λύσης (originalCells), τα οποία εμφανίζονται ως γεμισμένα (FILLED) ή κενά (EMPTY).
- Δεν αποδίδονται clues ή toolbars, παρά μόνο το τελικό οπτικό αποτέλεσμα.

Η λειτουργία αυτή δεν αποτελεί απαραίτητο μέρος της επίλυσης, αλλά ένα βοηθητικό εργαλείο ελέγχου και οπτικοποίησης. Σε Edit Mode, λειτουργεί ως τελική προεπισκόπηση πριν την αποθήκευση ενός επιπέδου. Σε Play Mode, μπορεί να χρησιμοποιηθεί ως βοήθημα ή επιβεβαίωση μετά την επίλυση.

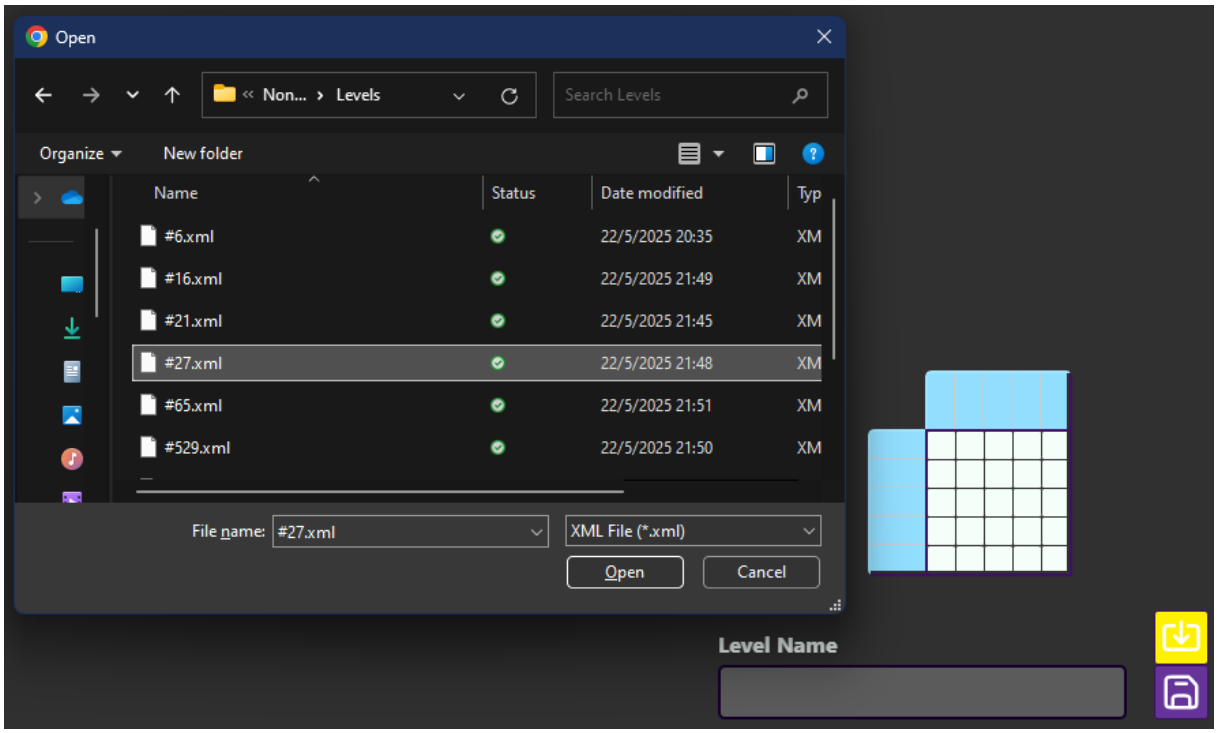
Η ενεργοποίηση της λειτουργίας δεν επηρεάζει την κατάσταση isSolved, ούτε αλλάζει τα δεδομένα του playingCells. Ο χρήστης μπορεί να εισέλθει και να εξέλθει από την προβολή preview χωρίς απώλεια δεδομένων ή αλλαγή στο progression.

Η παρουσίαση της σωστής εικόνας μέσα από τον ίδιο μηχανισμό rendering του NonogramTable εξασφαλίζει ότι η τελική εικόνα που βλέπει ο χρήστης είναι απόλυτα ακριβής, και παρέχει εγγύηση ότι η σύνθεση του επιπέδου συμφωνεί με τη λογική των clues.

3.9 Εισαγωγή Έτοιμων Επιπέδων

Η εφαρμογή Nonogram υποστηρίζει τη δυνατότητα εισαγωγής προκαθορισμένων επιπέδων μέσω αρχείων .xml, προσφέροντας στον χρήστη έναν εναλλακτικό τρόπο δημιουργίας γρίφων, χωρίς την ανάγκη χειροκίνητης σχεδίασης από την αρχή. Τα εισαγόμενα αρχεία ακολουθούν τη δομή του προτύπου **WebPBN** [4], ενός διαδικτυακού αποθετηρίου Nonogram puzzles, το οποίο επιτρέπει την εξαγωγή γρίφων σε μορφή αναγνώσιμη από εφαρμογές τρίτων.

Η λειτουργία εισαγωγής είναι ενσωματωμένη στο **Edit Mode** της εφαρμογής και υλοποιείται ως διακριτό κουμπί “Import” (κίτρινο background) στο κάτω μέρος της φόρμας αποθήκευσης. Με το πάτημα του κουμπιού, εμφανίζεται το σύστημα επιλογής αρχείου του browser, από όπου ο χρήστης μπορεί να επιλέξει ένα αρχείο .xml από το τοπικό του σύστημα αρχείων (Εικόνα 3.11). Δεν απαιτείται προηγούμενη ρύθμιση ή συμβατότητα αρχείου εκ μέρους του χρήστη, πέρα από τη βασική μορφή που παρέχει το WebPBN.



Εικόνα 3.13 Στιγμιότυπο από την εφαρμογή. Φαίνεται το κουμπί Import με κίτρινο background και το παράθυρο επιλογής αρχείου του browser.

Με την επιλογή αρχείου, ενεργοποιείται η συνάρτηση handleImportLevel() που ορίζεται στο Game.js. Η συνάρτηση αυτή χρησιμοποιεί το DOM API (DOMParser) για να μετατρέψει το περιεχόμενο του XML σε DOM αντικείμενο και να εξάγει τις βασικές πληροφορίες του επιπέδου (Εικόνα 3.13):

- **Τίτλος** (από το <title>),
- **Διαστάσεις** του grid **(**από τη δομή του <image> μέσα στο <solution>),
- **Λύση** του puzzle**,** η οποία βρίσκεται ως χαρακτήρες X και . σε κάθε γραμμή του <image>.

Η επεξεργασία κάθε γραμμής του πλέγματος περιλαμβάνει την αφαίρεση περιμετρικών χαρακτήρων |, τη συμπλήρωση με τελείες (.) όπου χρειάζεται για να διατηρηθεί σταθερό μήκος (padEnd), και τη μετατροπή κάθε χαρακτήρα σε boolean τιμή μέσω map(ch => ch === "X"). Το αποτέλεσμα είναι ένας πίνακας true/false που αποθηκεύεται στο state cells.

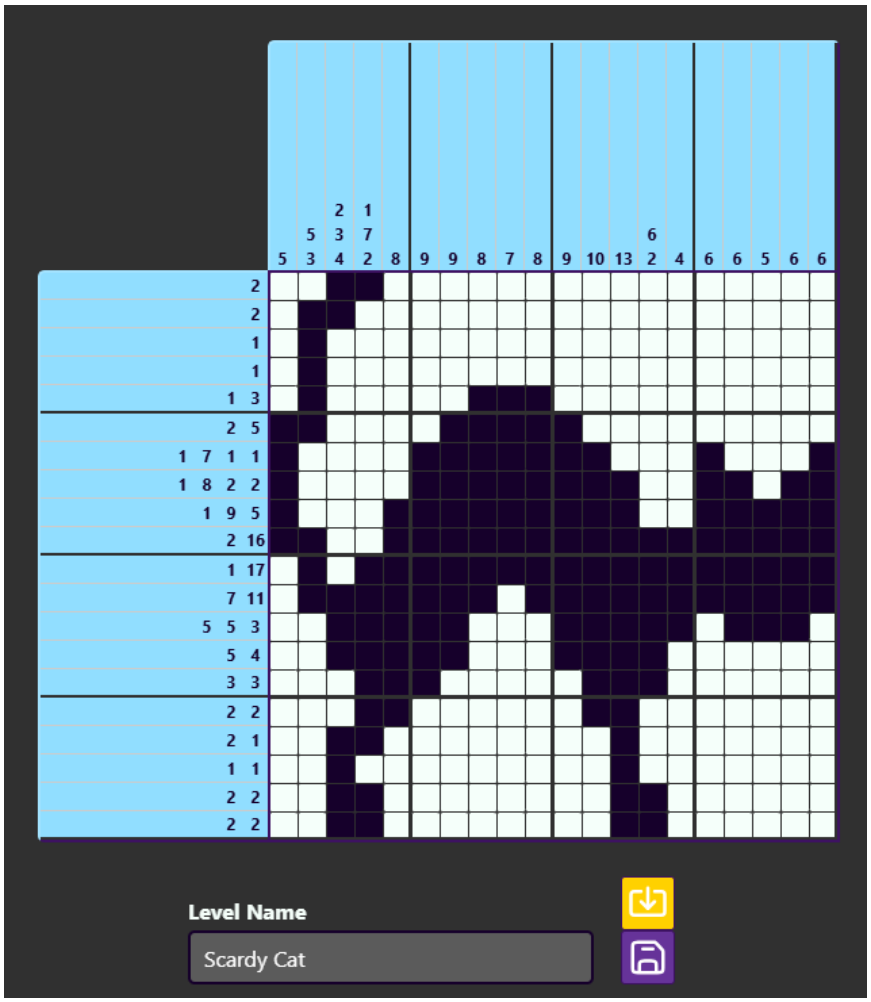
```
<title>Scardy Cat</title>
<solution type="goal">
<image>
|..XX..............|
|.XX...............|
|.X................|
|.X................|
|.X.....XXX........|
|XX....XXXXX.......|
|X....XXXXXXX...X..|
|X....XXXXXXXX..XX.|
|X...XXXXXXXXX..XXX|
|XX..XXXXXXXXXXXXX|
|.X.XXXXXXXXXXXXXX|
|.XXXXXXX.XXXXXXXXX|
|..XXXXX...XXXXX.XX|
|..XXXXX...XXXX....|
|...XXX.....XXX....|
|...XX......XX.....|
|..XX........X.....|
|..X.........X.....|
|..XX........XX....|
|..XX........XX....|
</image>
</solution>

</puzzle>
</puzzleset>
```

Εικόνα 3.14 Απόσπασμα του XML αρχείου που περιέχει τη λύση του επιπέδου

Μετά την εισαγωγή του πλέγματος:

- Το state size ενημερώνεται με τις σωστές διαστάσεις (width, height) ώστε να προσαρμοστεί το UI.
- Το levelName τίθεται με βάση τον τίτλο του puzzle, αν υπάρχει στο XML.
- Η συνάρτηση fillClues() καλείται αυτόματα ώστε να υπολογιστούν εκ νέου οι ενδείξεις του πλέγματος (rowClues, colClues), στηριζόμενη μόνο στο cells — χωρίς να γίνεται χρήση των clues που περιέχονται στο αρχείο XML.

Η εφαρμογή συνεχίζει να λειτουργεί όπως σε οποιοδήποτε manually σχεδιασμένο επίπεδο. Ο χρήστης μπορεί να τροποποιήσει τον γρίφο, να προσθέσει ή να αφαιρέσει κελία, να αλλάξει τον τίτλο και, όταν το επιθυμεί, να αποθηκεύσει το επίπεδο στη local IndexedDB μέσω Dexie. Η εισαγωγή μέσω XML δεν επηρεάζει τη δυνατότητα αποθήκευσης, ούτε επιβάλλει περιορισμούς στην περαιτέρω επεξεργασία.



Εικόνα 3.15 Στιγμιότυπο από την εφαρμογή. Φαίνεται το πλέγμα που προέκυψε μετά από επιτυχή εισαγωγή.

Σε επίπεδο εμπειρίας χρήστη, η διαδικασία είναι σχεδόν στιγμιαία και δεν απαιτεί ανανέωση ή επαναφόρτωση της σελίδας. Ο χρήστης βλέπει άμεσα το νέο πλέγμα, με γεμισμένα κελιά και υπολογισμένες ενδείξεις, έτοιμο προς επεξεργασία ή αποθήκευση. Αν το επιθυμεί, μπορεί στη συνέχεια να εισέλθει στο Play Mode για να επιχειρήσει την επίλυση του ίδιου γρίφου, χωρίς να χαθεί καμία πληροφορία από την εισαγωγή.

Η δυνατότητα αυτή ενισχύει σημαντικά την ευελιξία της εφαρμογής, καθώς επιτρέπει τη μεταφορά πλούσιου περιεχομένου από αποθετήρια όπως το WebPBN, τη συνεργασία με άλλους δημιουργούς γρίφων και τη γρήγορη αναπαραγωγή γρίφων για τεστ ή επίδειξη.

3.10 Αποθήκευση και Ανάκτηση Επιπέδων – IndexedDB και Dexie.js

Η δυνατότητα αποθήκευσης και ανάκτησης επιπέδων είναι καθοριστικής σημασίας για την εμπειρία χρήστη στην εφαρμογή Nonogram, καθώς επιτρέπει τη δημιουργία, αποθήκευση και επαναφόρτωση προσωπικών γρίφων. Η λειτουργία αυτή υλοποιείται αποκλειστικά μέσω client-side αποθήκευσης με χρήση της IndexedDB, αξιοποιώντας τη βιβλιοθήκη Dexie.js για μεγαλύτερη ευχρηστία και απλότητα στην υλοποίηση.

Η IndexedDB είναι μία εγγενώς ενσωματωμένη βάση δεδομένων σε όλους τους σύγχρονους browsers. Υποστηρίζει key-value structured αποθήκευση, indexing, querying και διαχείριση μεγάλου όγκου δεδομένων (π.χ. 50MB+ ανά domain). Σε αντίθεση με απλούστερες λύσεις όπως το localStorage, η IndexedDB είναι ασύγχρονη (non-blocking) και ιδανική για σύνθετα δεδομένα, κάτι που την καθιστά κατάλληλη για εφαρμογές τύπου puzzle editor όπου απαιτείται αποθήκευση πινάκων, clues και metadata [8].

Ωστόσο, το raw API της IndexedDB θεωρείται verbose και δύσχρηστο. Για τον λόγο αυτό, η εφαρμογή υλοποιεί τη διασύνδεση με τη βάση μέσω της Dexie.js – μιας βιβλιοθήκης JavaScript η οποία παρέχει abstraction layer και προσφέρει συντακτικό μοντέλο κοντά σε αυτό των παραδοσιακών relational βάσεων δεδομένων. Η Dexie υποστηρίζει promise-based async λειτουργία, schema versioning, transactional writes και indexes με απόδοση που δεν επηρεάζει το UI [6].

Η βάση ορίζεται στο context MainContext.js και περιλαμβάνει ένα object store με όνομα dbLevels, το οποίο αντιπροσωπεύει τα αποθηκευμένα επίπεδα του χρήστη. Κάθε επίπεδο περιέχει:

- **name**: Όνομα επιπέδου (string)
- **cells**: Grid αντικειμένων (Array of Arrays – κάθε cell ως object {state, x})
- **width**, **height**: Διαστάσεις του πλέγματος
- **createdAt**: Timestamp δημιουργίας (προσθήκη με Date.now())
- **solved**: boolean πεδίο για σήμανση επίλυσης

Κατά τη διαδικασία αποθήκευσης (μέσω handleSaveLevelClick() στο Game.js, βλ. Απόσπασμα Α.7 στο Παράρτημα Α), το grid επικυρώνεται και προστίθεται στην IndexedDB.

Η ανάκτηση των δεδομένων ενός επιπέδου γίνεται με χρήση της get μεθόδου από την διεπαφή της Dexie.js (βλ. Απόσπασμα Α.8 στο Παράρτημα Α).

Το αποτέλεσμα επιστρέφει αντικείμενο που μετατρέπεται σε cells, originalCells κ.λπ. Η μετατροπή Boolean σε αντικείμενα με state (π.χ. από true σε { state: FILLED, x: false }) γίνεται με χρήση χαρτογράφησης map(...) για να διασφαλιστεί η σωστή συμπεριφορά στο Play Mode.

Η προσβασιμότητα στη βάση επιτυγχάνεται μέσω του React Context API (useMainContext()), το οποίο παρέχει στα components τη μεταβλητή dbLevels και metadata όπως selectedLevel, size. Το πλεονέκτημα της λύσης αυτής είναι ότι η αποθήκευση, η φόρτωση και η ενημέρωση των επιπέδων γίνεται καθολικά, χωρίς prop drilling ή state duplication.

Η ανάκτηση όλων των επιπέδων γίνεται με με χρήση του hook useLiveQuery() της βιβλιοθήκης dexie-react-hooks και εμφανίζονται στο component Sidebar.js. Η κλήση:

```
const allLevels = useLiveQuery( querier: () => dbLevels.toArray(),  deps: []);
```

δημιουργεί ένα reactive observable stream που επαναφέρνει την τρέχουσα λίστα των επιπέδων σε κάθε αλλαγή στην IndexedDB. Αυτό επιτρέπει δυναμική ενημέρωση του UI χωρίς refresh.

Το component Sidebar.js παρουσιάζει όλα τα αποθηκευμένα επίπεδα ομαδοποιημένα ανά διάσταση (10x10, 15x15 κ.λπ.). Η λογική αυτή προσφέρει στον χρήστη μια οργανωμένη λίστα γρίφων με δυνατότητα άμεσης επιλογής, και ανακατεύθυνση στο Game view με χρήση navigate(...). Η λειτουργία

handleSelectLevelClick(...) ορίζει το selectedLevel και το μέγεθος (setSize(...)), δημιουργώντας μία seamless εμπειρία περιήγησης.

Η χρήση του useLiveQuery() εξασφαλίζει ότι το sidebar ενημερώνεται αυτόματα όταν προστίθεται, αφαιρείται ή τροποποιείται ένα επίπεδο. Αυτό αφαιρεί την ανάγκη για manual refresh και ενισχύει τη δυναμική αίσθηση της εφαρμογής. Το hook είναι ιδανικό για real-time rendering και παρέχει βελτιστοποιημένη performance με ελάχιστο re-rendering.

Λειτουργίες της Dexie.js

Η Dexie υποστηρίζει επιπλέον λειτουργίες όπως:

- Indexing για πεδία όπως name, createdAt, solved,
- Query chains, π.χ. dbLevels.where("solved").equals(true),
- Versioning για μελλοντικές αλλαγές στο schema.

Μπορεί να επεκταθεί ώστε να παρέχει sorting (κατά όνομα, ημερομηνία), φίλτρα ανά solved/unresolved, grouping και export λειτουργίες σε JSON.

Συγκριτικά με εναλλακτικές τεχνολογίες:

- **localStorage:** υποστηρίζει μόνο string storage, δεν είναι async, περιορίζεται σε λίγα KB.
- **sessionStorage:** ίδια όρια με localStorage, αλλά διαγράφεται με την αποσύνδεση του tab.
- **WebSQL:** deprecated και μη υποστηριζόμενο από όλους τους browsers.

Άρα, η επιλογή Dexie + IndexedDB αποτελεί βέλτιστη λύση για μοντέρνες SPA εφαρμογές που απαιτούν data persistence, ακόμα και offline.

Τέλος, η αποθήκευση επιπέδων στον browser έχει άμεσο θετικό αντίκτυπο στην εμπειρία χρήστη, καθώς:

- Δίνει την αίσθηση ιδιοκτησίας και προόδου (user-created content),
- Επιτρέπει offline επίλυση και επιστροφή σε προηγούμενα puzzles,
- Ενισχύει το interaction και τη διάδραση.

Η ευελιξία στην αρχιτεκτονική επιτρέπει μελλοντική επέκταση σε εξαγωγή επιπέδων, αποστολή σε backend ή κοινή χρήση.

3.11 Βοηθητικές Συναρτήσεις και Υποστηρικτική Λογική

Η συνολική λειτουργικότητα της εφαρμογής Nonogram δεν βασίζεται μόνο στα βασικά components (όπως το Game.js και το NonogramTable.js), αλλά και σε ένα σύνολο κρίσιμων βοηθητικών συναρτήσεων και μηχανισμών υποστήριξης. Οι συναρτήσεις αυτές αναλαμβάνουν ειδικούς υπολογισμούς, ενδιάμεσες επεξεργασίες και ενέργειες υποδομής που καθιστούν εφικτή την ορθή και απρόσκοπτη λειτουργία του παιχνιδιού.

Από την αυτόματη παραγωγή ενδείξεων (clues), μέχρι τη σήμανση ολοκληρωμένων clues, την υποστήριξη auto-fill και την κατανόηση των ενεργειών του χρήστη, κάθε βοηθητική συνάρτηση διαδραματίζει καθοριστικό ρόλο. Η ύπαρξή τους ενισχύει τη modular αρχιτεκτονική της εφαρμογής και επιτρέπει την εύκολη επεκτασιμότητα, απομόνωση λογικής και την αποδοτική συντήρηση.

Στις υποενότητες που ακολουθούν, παρουσιάζονται οι σημαντικότερες από αυτές τις συναρτήσεις, η λογική τους, η εφαρμογή τους και ο τρόπος με τον οποίο ενσωματώνονται στη λειτουργία του Nonogram.

3.11.1 Παραγωγή Ενδείξεων (Clues) από Πλέγμα

Η παραγωγή των ενδείξεων (clues) γίνεται με την συνάρτηση fillClues(grid). Είναι υπεύθυνη για τη δημιουργία των αριθμητικών ενδείξεων (clues) που αντιστοιχούν σε κάθε γραμμή και στήλη του πλέγματος. Οι ενδείξεις αυτές αποτελούν το μοναδικό μέσο πληροφορίας που έχει ο παίκτης για να επιλύσει τον γρίφο. Η παραγωγή τους βασίζεται στην καταμέτρηση συνεχόμενων γεμισμένων κελιών (FILLED) εντός κάθε γραμμής ή στήλης.

Η λογική της fillClues() (βλ. Απόσπασμα Α.9 στο Παράρτημα Α) εφαρμόζεται τόσο κατά τη δημιουργία επιπέδου (Edit Mode), όσο και κατά την προεπισκόπηση ή τη φόρτωση grid σε Preview Mode. Εσωτερικά, η συνάρτηση ορίζει μία βοηθητική getLineClues(line) η οποία δέχεται ως είσοδο έναν πίνακα κελιών (γραμμή ή στήλη) και επιστρέφει πίνακα αντικειμένων clues.

Η λειτουργία έχει ως εξής:

1. Διατρέχει το κάθε κελί της γραμμής.
2. Αν το κελί είναι γεμισμένο (cell.state === FILLED), αυξάνει έναν μετρητή.
3. Αν βρει κενό κελί και ο μετρητής > 0, δημιουργεί clue με το πλήθος και τον μηδενίζει.
4. Αν υπάρχει μετρητής στο τέλος της γραμμής, τον προσθέτει ως clue.
5. Αν δεν εντοπιστούν γεμισμένα κελιά, επιστρέφει κενό πίνακα.

Παράδειγμα εισόδου:

[{state: 0}, {state: 1}, {state: 1}, {state: 0}, {state: 1}]

Παραγόμενη έξοδος:

[{done: false, value: 2}, {done: false, value: 1}]

Αφού παραχθούν τα clues για όλες τις γραμμές (tempRowClues = grid.map(getLineClues)), η συνάρτηση δημιουργεί τις στήλες μέσω μετασχηματισμού (transpose) του πίνακα grid:

```
const transposedGrid = grid[0].map((_, i) => grid.map((row) => row[i]));
```

Έπειτα εφαρμόζει την ίδια getLineClues() στις στήλες για παραγωγή tempColClues.

Τέλος, καλεί τις setRowClues() και setColClues() για να ενημερώσει την κατάσταση της εφαρμογής και να αποδώσει τις ενδείξεις στο UI. Η δομή κάθε clue είναι αντικείμενο με value (το πλήθος κελιών) και done (αν η ένδειξη έχει καλυφθεί κατά την επίλυση).

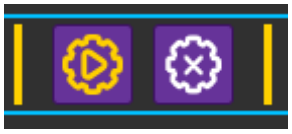
Η χρήση αυτής της συνάρτησης είναι θεμελιώδης για την ορθή απόδοση και λογική συνέπεια του Nonogram. Κάθε αλλαγή στο πλέγμα (π.χ. κατά τη σχεδίαση) απαιτεί ανανέωση των clues ώστε να αποτυπώνεται η ακριβής δομή του επιπέδου.

3.11.2 Αυτόματη Σήμανση Λυμένων Clues

Η συνάρτηση markCluesAsDone(grid) αποτελεί ένα κρίσιμο εργαλείο που υποστηρίζει τη διαδραστικότητα και την αυτοματοποίηση της εμπειρίας χρήστη στην εφαρμογή Nonogram. Η βασική της λειτουργία είναι να αναλύει την τρέχουσα κατάσταση του πλέγματος (grid) και να προσδιορίζει ποιες ενδείξεις (clues) έχουν καλυφθεί επιτυχώς από τον παίκτη. Η πληροφορία αυτή καταγράφεται σε κάθε clue με την ιδιότητα done: true, παρέχοντας οπτική επιβεβαίωση ότι η συγκεκριμένη γραμμή ή στήλη έχει λυθεί σωστά.

Η ανάγκη για αυτόματη σήμανση προκύπτει από την εγγενή φύση των Nonogram: ο παίκτης χρησιμοποιεί μόνο τις ενδείξεις για να καταλήξει σε μία λύση. Καθώς δεν υπάρχει άμεση επιβεβαίωση αν οι επιλογές του είναι σωστές, η λειτουργία autoMarkClues έρχεται να καλύψει αυτό το κενό, παρέχοντας διακριτική αλλά χρήσιμη πληροφόρηση. Η markCluesAsDone καλείται κάθε φορά που τροποποιείται το πλέγμα (συνήθως κατά το mouseUp) και είναι ενεργοποιημένη η ρύθμιση autoMarkClues στο Play Mode.

Η ενεργοποίηση της αυτόματης σήμανσης λυμένων clues γίνεται με την επιλογή του πλήκτρου με το Play εικονίδιο μέσα σε ένα γρανάζι, από το Toolbar μενού. Θα φανεί ότι είναι ενεργοποιημένο με ένα highlight με κίτρινο στο εσωτερικό του πλήκτρου, όπως φαίνεται στην παρακάτω εικόνα. Παρόμοια ενεργοποιείται και η λειτουργία αυτόματης συμπλήρωσης με X, πατώντας το πλήκτρο με το εικονίδιο X μέσα σε ένα γρανάζι. Η λειτουργία αυτόματης συμπλήρωσης με X παρουσιάζεται στην επόμενη ενότητα.



Εικόνα 3.16 Στιγμιότυπο από την εφαρμογή. Αυτόματη σήμανση λυμένων Clues (αριστερό κουμπί) και Αυτόματη Συμπλήρωση με X (δεξί κουμπί).

Η ανάλυση πραγματοποιείται ξεχωριστά για κάθε γραμμή και στήλη. Ο αλγόριθμος αποτελείται από δύο κύρια στάδια: πρώτον, την απομόνωση γεμισμένων ακολουθιών (FILLED segments) σε κάθε γραμμή ή στήλη και, δεύτερον, τη σύγκριση των ακολουθιών αυτών με τις τιμές των clues. Αν ο αριθμός και η σειρά των ακολουθιών συμφωνεί με τα clues, τότε θεωρείται ότι η γραμμή ή στήλη έχει λυθεί σωστά.

Αναλυτικά, η βοηθητική συνάρτηση checkClueDone(line, clues) (βλ. Απόσπασμα Α.10 στο Παράρτημα Α) που εκτελεί τον έλεγχο περιλαμβάνει τα εξής βήματα:

1. Αρχικοποιείται ένας μετρητής filledCount = 0 και ένας δείκτης clueIndex = 0.
2. Δημιουργείται πίνακας matches ίδιου μήκους με τα clues, αρχικά γεμάτος με false.
3. Διατρέχουμε κάθε κελί της γραμμής. Για κάθε κελί:
    - Αν είναι γεμισμένο (cell.state === FILLED), αυξάνεται ο filledCount.

- Αν είναι κενό και ο filledCount > 0, ελέγχεται αν ο τρέχων clue.value ισούται με filledCount. Αν ναι, θέτουμε matches[clueIndex] = true.
- Μηδενίζουμε τον filledCount και προχωράμε στον επόμενο clueIndex.

4. Αν στο τέλος του line υπάρχει υπόλοιπο filledCount, ελέγχεται αν ταιριάζει με το τελευταίο clue.
5. Επιστρέφεται ο πίνακας matches, που δείχνει ποια clues έχουν επιτευχθεί.

Η markCluesAsDone() καλεί την checkClueDone(...) για όλες τις γραμμές (rowClues) και όλες τις στήλες, αφού πρώτα γίνει transposition του grid. Οι τιμές matches[] χρησιμοποιούνται για να δημιουργηθούν νέα clues με τις υπάρχουσες value και νέο πεδίο done: true/false αντίστοιχα. Οι ενημερωμένοι πίνακες clues αποθηκεύονται μέσω των setRowClues() και setColClues().

Στο interface, τα clues με done: true αποδίδονται οπτικά μέσω CSS (nonogramtable__clues__clue--done), παρέχοντας άμεση επιβεβαίωση στον χρήστη χωρίς να παραβιάζεται η λογική πρόκληση του puzzle (βλ. Εικόνα 3.17). Ο παίκτης βλέπει ποια clues έχει καλύψει σωστά, ακόμα και χωρίς να χρησιμοποιήσει λειτουργίες ελέγχου λύσης.

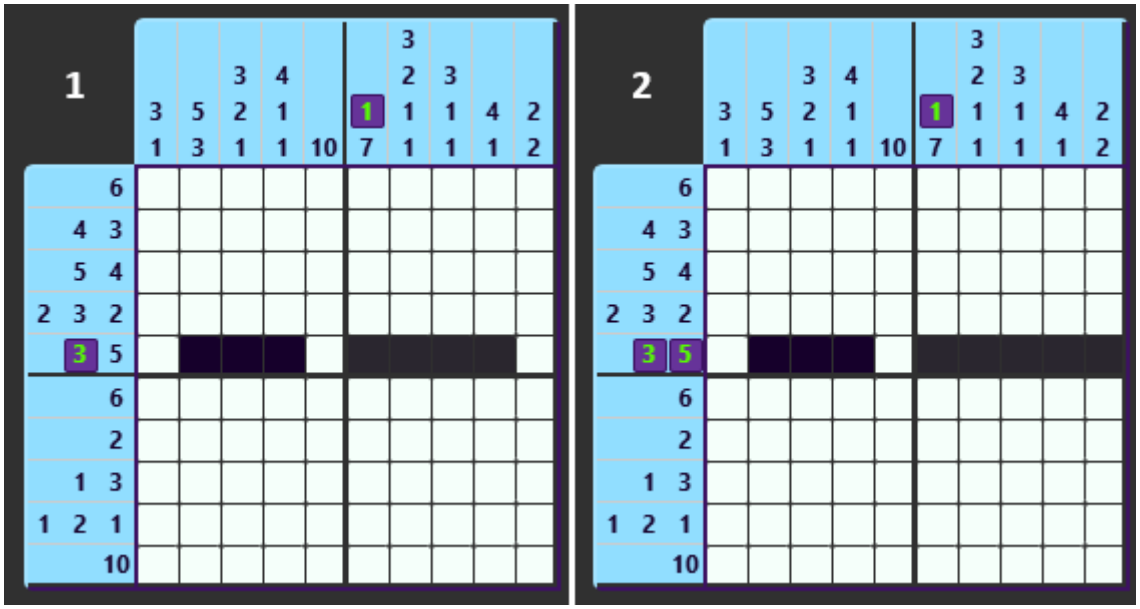
Παράδειγμα (βλ. Εικόνα 3.14):

Έστω γραμμή με clues [3, 5] και grid:

[_ ● ● ● _ ● ● ● ● ●]

Η checkClueDone επιστρέφει [true, true], και οι clues αποθηκεύονται ως:

[{value: 3, done: true}, {value: 5, done: true}]



Εικόνα 3.17 Στιγμιότυπα από την εφαρμογή. Φαίνεται η σήμανση του clue [3, 5] πριν (1) και μετά (2) αφού συμπληρωθεί και το 5ο συνεχόμενο κελί στην γραμμή.

Αξίζει να σημειωθεί ότι ο έλεγχος είναι καθαρά αριθμητικός και δεν αξιολογεί την ύπαρξη περιττών γεμισμένων κελιών εκτός των συμφωνημένων clues. Αυτό εξασφαλίζει ταχύτητα και απλότητα, αφήνοντας τον τελικό έλεγχο μοναδικότητας και εγκυρότητας στη συνάρτηση handleCheckSolvedClick().

Επιπρόσθετα, η πληροφορία done: true αξιοποιείται και από τη συνάρτηση applyAutoX(grid), η οποία γεμίζει αυτόματα με X όλα τα υπόλοιπα κελιά της γραμμής ή στήλης, όταν τα clues έχουν ολοκληρωθεί. Έτσι, δημιουργείται μία λειτουργική αλυσίδα μεταξύ markClues → autoX → visual feedback.

Η συνάρτηση είναι επεκτάσιμη για μελλοντικές προσθήκες, όπως παρακολούθηση ποσοστού επίλυσης, ενεργοποίηση συμβουλών μόνο σε μη επιλυμένα clues, ή αξιολόγηση δυσκολίας με βάση την πρόοδο στις clues. Η υλοποίησή της είναι modular και πλήρως ενσωματωμένη στο rendering σύστημα της εφαρμογής.

3.11.3 Αυτόματη Συμπλήρωση με X

Η συνάρτηση applyAutoX(grid) προσφέρει στον παίκτη μία λειτουργία αυτόματης σημείωσης (marking) των κελιών που θεωρούνται άδεια (λευκά), με χρήση του συμβόλου X. Αυτή η δυνατότητα ενεργοποιείται όταν είναι επιλεγμένη η ρύθμιση autoAddX στο Play Mode και βασίζεται στη λογική ότι μόλις όλα τα clues μιας γραμμής ή στήλης έχουν ολοκληρωθεί (δηλαδή έχουν γίνει done: true μέσω της markCluesAsDone), τότε τα υπόλοιπα μη γεμισμένα κελιά μπορούν να θεωρηθούν άδεια και να σημειωθούν αυτόματα με X.

Η λειτουργία αυτή διευκολύνει τον παίκτη στην πλοήγηση μέσα σε μεγάλα ή σύνθετα πλέγματα, καθώς εξοικονομεί χρόνο και μειώνει την οπτική φόρτιση του grid. Παράλληλα, ελαχιστοποιεί την πιθανότητα να ξανα-εξεταστούν ήδη λυμένες περιοχές, ενισχύοντας την εμπειρία και την απόδοση.

Όπως αναφέρθηκε και στην προηγούμενη ενότητα, η ενεργοποίηση της αυτόματης σημπλήρωσης με X γίνεται με την επιλογή του πλήκτρου με το X εικονίδιο μέσα σε ένα γρανάζι, από το Toolbar μενού. Με την ενεργοποίησή της, ενεργοποιηθείται αυτόματα και η αυτόματη συμπλήρωση λυμένων clues καθώς εξαρτάται από το αν είναι λυμένα όλα τα clues.

Ο αλγόριθμος της applyAutoX(grid), που παρουσιάζεται αναλυτικά στο Παράρτημα Α.12, εφαρμόζεται ως εξής:

1. Δημιουργείται νέο grid με deep copy από το αρχικό (updated = grid.map(...)) ώστε να μην επηρεάζεται απευθείας το state.
2. Για κάθε γραμμή (row), ελέγχεται αν τα clues της έχουν done: true.
    - Αν ναι, τότε κάθε κελί που δεν είναι γεμισμένο (state !== FILLED) αντικαθίσταται με **{state: EMPTY, x: true}.**
3. Αντίστοιχα για κάθε στήλη (col), ελέγχεται αν οι clues έχουν επιλυθεί.
    - Αν ναι, για κάθε κελί της στήλης που δεν είναι γεμισμένο, προστίθεται x: true.
4. Το νέο grid επιστρέφεται και αντικαθιστά το state μέσω setCells(...) ή setPlayingCellsWithHistory(...).

Η συνάρτηση βασίζεται στις δομές clues (rowClues, colClues) που περιέχουν το πεδίο done. Το done παράγεται από την markCluesAsDone(grid). Άρα, η χρήση της applyAutoX προϋποθέτει ότι ο μηχανισμός σήμανσης έχει προηγηθεί.

Παράδειγμα (βλ. Εικόνα 3.16):

Γραμμή clues: [5]

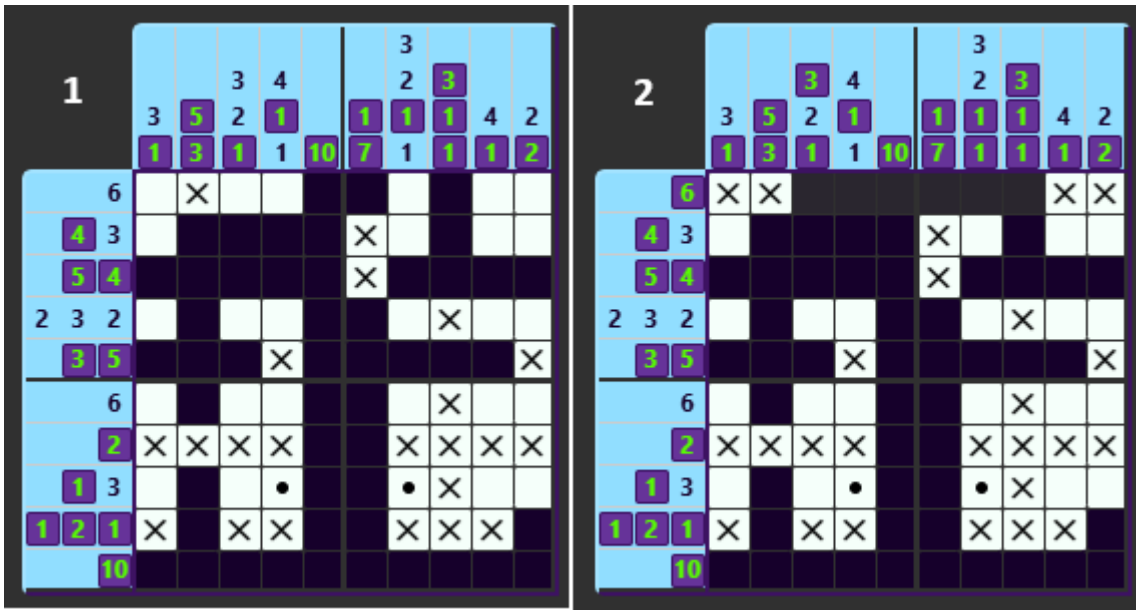
Grid: [□ □ ● ● ● ● ● ● □ □]

Αν η σειρά ● ● ● ● ● ● καλύπτει την ένδειξη 6 και η υπόλοιπη γραμμή είναι άδεια, η applyAutoX θα μετατρέψει τα υπόλοιπα κελιά σε:

{ state: EMPTY, x: true }

και το αποτέλεσμα στο UI θα είναι:

[X X ● ● ● ● ● ● X X]



Εικόνα 3.18 Στιγμιότυπα από την εφαρμογή. Φαίνεται η κατάσταση του πλέγματος πριν (1) και μετά (2) από το γέμισμα των 6 κελιών της 1ης γραμμής, όπου και γεμίζουν αυτόματα τα υπόλοιπα της σειράς με X.

Η συνάρτηση ενσωματώνεται στο component NonogramTable.js και καλείται μέσα από την fillCell(...) όταν αλλάζει η κατάσταση του grid, υπό την προϋπόθεση ότι το autoAddX είναι ενεργό. Ενδείκνυται σε συνδυασμό με autoMarkClues για συνεπή εμπειρία.

Από σχεδιαστικής άποψης, το X δεν επηρεάζει την εγκυρότητα ή την επίλυση του puzzle — είναι καθαρά visual marker. Ο μηχανισμός επίλυσης (handleCheckSolvedClick) δεν εξετάζει το πεδίο x, διατηρώντας τον διαχωρισμό λογικής και αναπαράστασης.

Η applyAutoX μπορεί να επεκταθεί στο μέλλον ώστε να:

- Εφαρμόζει X μόνο όταν η γραμμή/στήλη έχει επίσης isSolved === true.
- Καταγράφει αριθμό ενεργειών X για στατιστική παρακολούθηση.
- Επιτρέπει επιλογή μοτίβου ή συμβόλου αντί για X.

3.11.4 Εντοπισμός Μη Σημπληρωμένων Τμημάτων

Η συνάρτηση getUnfilledSegments(originalGrid, playingGrid) (βλ. Απόσπασμα Α.13 στο Παράρτημα Α) αποτελεί βασικό υποσύστημα λογικής της εφαρμογής Nonogram και χρησιμοποιείται κυρίως από το Hint Engine για τον εντοπισμό εκείνων των περιοχών του grid όπου ο χρήστης δεν έχει ολοκληρώσει ακόμα τις αναμενόμενες γεμισμένες ακολουθίες. Η λειτουργία της επιτρέπει στην εφαρμογή να προτείνει έγκυρες υποδείξεις που σχετίζονται με τη λογική επίλυσης και δεν βασίζονται σε τυχαία κριτήρια.

Η λειτουργία της βασίζεται σε μια σύγκριση ανάμεσα στο grid της λύσης (originalGrid) και στο πλέγμα του χρήστη (playingGrid). Για κάθε γραμμή και στήλη εντοπίζονται τα segments — δηλαδή οι συνεχόμενες ακολουθίες κελιών που πρέπει να είναι γεμισμένα — και εξετάζεται αν έχουν συμπληρωθεί. Αν δεν έχουν, προστίθενται στη λίστα των υποψηφίων τμημάτων για υπόδειξη.

Αναλυτικά η λογική του αλγορίθμου περιλαμβάνει τα εξής (Εικόνα 3.19):

1. Εντοπισμός τμημάτων (segments) στο originalGrid:
    - Για κάθε γραμμή, διατρέχονται τα κελιά και αναγνωρίζονται σειρές συνεχόμενων κελιών με state === FILLED.
    - Το σημείο εκκίνησης (start) και λήξης (end) κάθε τέτοιου block καταγράφονται.
2. Έλεγχος πληρότητας στο playingGrid:
    - Για κάθε segment που εντοπίστηκε στο originalGrid, η συνάρτηση εξετάζει αν τα αντίστοιχα κελιά στο playingGrid έχουν και αυτά state === FILLED.
    - Αν οποιοδήποτε από τα κελιά δεν είναι σωστά συμπληρωμένο, θεωρείται ότι το segment είναι "ανολοκλήρωτο".
3. Επιστροφή δεδομένων:
    - Κάθε μη συμπληρωμένο segment αποθηκεύεται ως αντικείμενο:
    - **{ type: 'row' | 'col', index: number, start: number, end: number }**

Ο ίδιος αλγόριθμος εφαρμόζεται και για τις στήλες, με παρόμοια λογική. Στην περίπτωση των στηλών, η προσπέλαση είναι κατακόρυφη και το checking γίνεται στη μορφή grid[row][col], όπου col παραμένει σταθερό.

Το τελικό αποτέλεσμα της συνάρτησης είναι ένας πίνακας segments[], ο οποίος στη συνέχεια χρησιμοποιείται από το σύστημα υποδείξεων (handleHintClick) για να επιλεγεί πού πρέπει να εφαρμοστεί βοήθεια. Προτεραιότητα δίνεται συνήθως στο πιο πρόσφατα αγγιγμένο segment (lastFilledCell) ή σε εκείνα με μεγαλύτερο μήκος.

Η συνάρτηση λειτουργεί εντελώς αποσυνδεδεμένα από clues ή visual feedback, γεγονός που την καθιστά πολύτιμη για επεκτάσεις σε:

- προτάσεις επίλυσης βάσει στρατηγικής,
- βελτίωση της υποδειγματικής εμπειρίας (hint suggestions),
- δυνατότητες animation ή step-by-step εκπαίδευσης.

Η getUnfilledSegments(...) ενισχύει τη λογική συνοχή της εφαρμογής, διασφαλίζει σχετικότητα μεταξύ input και βοήθειας, και προσφέρει καθαρή επεκτασιμότητα στο Hint Engine χωρίς να επηρεάζει άλλες βασικές λειτουργίες του solving.

3.11.5 Ενημέρωση Κελιού με Τιμή

Για την ενημέρωση του Κελιού με Τιμή, όπως έχει αναφερθεί και προηγουμένως, γίνεται με την συνάρτηση fillCell(row, col, value) (βλ. Απόσπασμα Α.11 στο Παράρτημα Α). Είναι μία κεντρική βοηθητική συνάρτηση του component NonogramTable.js, υπεύθυνη για την καταγραφή μιας αλλαγής στο grid, όταν ο χρήστης επεμβαίνει με το ποντίκι σε κάποιο κελί. Είναι η συνάρτηση που εκτελείται κάθε φορά που ο χρήστης κάνει mouse down ή mouse enter (με πατημένο κλικ) σε ένα κελί. Εφαρμόζεται τόσο στο Edit Mode όσο και στο Play Mode, με διαφορετική μορφή της μεταβιβαζόμενης τιμής. Αν και είναι εννοιολογικά απλή, η σωστή της υλοποίηση διασφαλίζει τη σταθερότητα της κατάστασης του πλέγματος και την ορθότητα των λειτουργιών undo/redo και clues.

Η συνάρτηση ορίζεται στο component NonogramTable.js. Η fillCell δεν ενημερώνει απευθείας το props.cells, αλλά το sessionCells. Το sessionCells είναι ένα προσωρινό πλέγμα που διατηρείται μόνο κατά την περίοδο μιας ενέργειας (π.χ. drag session), επιτρέποντας ομαλή επεξεργασία πολλαπλών κελιών χωρίς συνεχές re-render. Η μεταφορά του session στο κανονικό grid γίνεται με setCells(sessionCells) όταν σηκωθεί το κουμπί του ποντικιού (mouse up).

Αναλυτική περιγραφή

1. **Η συνάρτηση καλείται με:**
    - **row:** ο δείκτης γραμμής του κελιού
    - **col:** ο δείκτης στήλης
    - **value:** η νέα τιμή που θα αποδοθεί στο κελί. Αυτή έχει διαφορετική μορφή ανάλογα με το mode:
        - **boolean** (true ή false) στο Edit Mode
        - **object** με δομή {state: 0|1|2, x: true|false} στο Play Mode
2. **Αντιγραφή γραμμής:** Η γραμμή του κελιού που θα αλλάξει αντιγράφεται (shallow copy) για να τροποποιηθεί χωρίς παρενέργειες:
3. **Ενημέρωση κελιού:** Η νέα τιμή αποδίδεται στο κελί μέσω της εντολής **newRow[col] = value;**
4. **Ανακατασκευή του πλέγματος:** Η μεταβλητή **updated** παίρνει το τρέχον **session,** με τη νέα γραμμή να αντικαθιστά την παλιά.

5. **Ανανέωση clues και X (αν ενεργό):** Αν είναι ενεργό το autoMarkClues, η markCluesAsDone() καλείται για να ενημερώσει το UI. Αν και το autoAddX είναι επίσης ενεργό, εφαρμόζεται και το applyAutoX() για τα αντίστοιχα clues.
6. **Καταγραφή συντεταγμένων:** Η θέση [row, col] του κελιού προστίθεται στο sessionCellCoords για να παρακολουθείται τι τροποποιήθηκε κατά το drag.

Παραδείγματα

Παράδειγμα 1 – Εισαγωγή γεμισμένου κελιού (Play Mode)

fillCell(3, 5, { state: 1, x: false });

Το κελί στη θέση [3][5] λαμβάνει κατάσταση FILLED, χωρίς να έχει σημειωθεί X. Η αλλαγή είναι οπτικά ένα μαύρο τετράγωνο.

Παράδειγμα 2 – Εισαγωγή κελιού με X (Play Mode)

fillCell(2, 7, { state: 0, x: true });

Το κελί [2][7] δεν είναι γεμισμένο, αλλά έχει επισημανθεί από τον χρήστη ως άκυρο (με X). Αυτό δεν επηρεάζει την εγκυρότητα της λύσης αλλά βοηθά τη λογική του παίκτη.

Παράδειγμα 3 – Toggle Boolean (Edit Mode)

fillCell(0, 0, true);

Σε Edit Mode, το κελί [0][0] μετατρέπεται σε γεμισμένο (true). Η ίδια κλήση με false θα το έκανε ξανά κενό.

Παράδειγμα 4 – Μαζική εισαγωγή (drag)

Όταν ο χρήστης κρατά πατημένο το αριστερό κουμπί και σύρει πάνω από [4][2] ως [4][7], η fillCell() θα κληθεί πολλές φορές με:

fillCell(4, 2, { state: 1, x: false });

fillCell(4, 3, { state: 1, x: false });

...

fillCell(4, 7, { state: 1, x: false });

Αυτό επιτρέπει σχεδίαση συνεχόμενων κελιών.

Σχέση με clues και auto-X

Η συνάρτηση είναι στενά συνδεδεμένη με τα εργαλεία σχεδίασης (drawState, drawMode) και τις αντίστοιχες συναρτήσεις handleMouseDown και handleMouseEnter. Αποτελεί τη βάση όλων των

αλληλεπιδράσεων με το grid και είναι θεμελιώδης για τη λειτουργία undo/redo, καθώς επεξεργάζεται δεδομένα εντός της περιόδου drag, και όχι απευθείας στο κύριο state.

Εφόσον χρησιμοποιείται μαζί με προσωρινή κατάσταση (sessionCells), αποτρέπει προβλήματα όπως τμηματική ενημέρωση κελιών, ακούσιες εγγραφές και re-rendering. Έχει επίσης ρόλο στην ανανέωση clues μέσω autoMarkClues και ενδέχεται να ενεργοποιήσει autoX όταν οι clues ολοκληρωθούν.

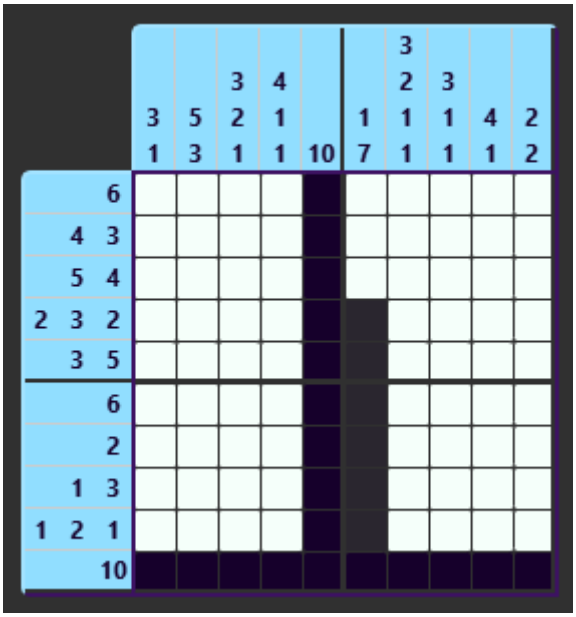
3.11.6 Χειρισμός Drag και sessionCells

Η υποστήριξη drag-and-draw στο Nonogram επιτρέπει στον χρήστη να γεμίσει ή να σημειώσει πολλαπλά κελιά με μία συνεχή ενέργεια (drag). Για την υλοποίηση αυτής της δυνατότητας, η εφαρμογή χρησιμοποιεί έναν συνδυασμό μηχανισμών: το προσωρινό πλέγμα sessionCells για ασφαλή επεξεργασία κατά τη διάρκεια της ενέργειας, και το drawMode, το οποίο καθορίζει αν η σχεδίαση γίνεται ελεύθερα ή με ευθυγράμμιση.

Η ύπαρξη του sessionCells καθιστά δυνατή την προσωρινή αποθήκευση των αλλαγών πριν αυτές εφαρμοστούν στο κύριο grid. Με αυτόν τον τρόπο, η εφαρμογή αποφεύγει συνεχόμενες ενημερώσεις της κατάστασης (state) σε κάθε πέρασμα του ποντικιού από διαφορετικά κελιά, γεγονός που βελτιώνει την απόδοση και αποτρέπει προβλήματα συγχρονισμού. Αντί να αποθηκεύεται κάθε κίνηση απευθείας, καταγράφονται όλες σε αυτό το buffer και εφαρμόζονται μαζικά όταν ολοκληρωθεί η ενέργεια.

Κατά την εκκίνηση της σχεδίασης με το onMouseDown, καταγράφεται η αρχική κατάσταση του πλέγματος με deep copy του props.cells, δημιουργώντας το sessionCells. Επίσης αποθηκεύεται η θέση έναρξης (drawStart) για χρήση σε LINE mode, καθώς και το είδος της ενέργειας (drawAction), δηλαδή αν αφορά την αλλαγή state ή την προσθήκη x. Επιπλέον, ενεργοποιείται το flag isDrawing.

Στη συνέχεια, κάθε φορά που το ποντίκι περνά πάνω από νέο κελί (onMouseEnter(row, col)), καλείται η fillCell(...), η οποία εφαρμόζει την κατάλληλη τιμή στο κελί μέσα στο sessionCells. Παράλληλα, καταγράφεται το row-col key στο sessionCellCoords ώστε να παρακολουθούμε ποια κελιά έχουν αλλαχθεί. Τα κελιά τα οποία έχουν αποθηκευτεί στο sessionCells σημάνονται με αντίστοιχο χρώμα (opacity 0.2 του αρχικού χρώματος) όπως φαίνεται στην παρακάτω εικόνα.



Εικόνα 3.19 Στιγμιότυπο από την εφαρμογή. Φαίνεται η χαρακτηριστική σήμανση με opacity 0.2 των κελιών sessionCells της 6ης στήλης.

Όταν το ποντίκι σηκωθεί (window.onMouseUp), ολοκληρώνεται το draw session και η εφαρμογή:

- μεταφέρει τα δεδομένα από το sessionCells στο κανονικό grid (setCells ή setPlayingCellsWithHistory),
- καθαρίζει όλα τα προσωρινά flags (sessionCellCoords, isDrawing, drawAction),
- και αποτρέπει επιπλέον ενημερώσεις μέχρι νέο mouseDown.

Ο συνδυασμός των παραπάνω εξασφαλίζει:

- Οπτική συνέπεια (δεν ανανεώνεται το UI σε κάθε κελί ξεχωριστά),
- Υποστήριξη undo σε επίπεδο ενέργειας (μία drag = μία εγγραφή στο stack),
- Ομαλή και σταθερή συμπεριφορά ανεξάρτητα από την ταχύτητα χρήσης.

Λειτουργία drawMode: FREE και LINE σχεδίαση

Το drawMode είναι μια παράμετρος που καθορίζει πώς αποδίδεται η σχεδίαση των κελιών από τον χρήστη. Διακρίνονται δύο καταστάσεις:

- **drawMode = 0 → FREE**: κάθε πέρασμα του ποντικιού επηρεάζει ανεξάρτητα το αντίστοιχο κελί.
- **drawMode = 1 → LINE**: κατά το drag σχεδιάζεται μία ολόκληρη γραμμή μεταξύ της θέσης εκκίνησης (drawStart) και της τρέχουσας θέσης.

Στην πράξη, κατά την είσοδο σε ένα νέο κελί, το handleMouseEnter καλεί είτε τη fillCell(...) (FREE mode), είτε τη handleLineDrawing() (LINE mode). Η handleLineDrawing() ελέγχει αν η κίνηση του χρήστη είναι ευθυγραμμισμένη σε γραμμή ή στήλη και γεμίζει όλα τα ενδιάμεσα κελιά που βρίσκονται στην ίδια row ή col.

Παράδειγμα 1 (FREE mode):

Ο χρήστης ξεκινά από [2][2] και περνά διαδοχικά από [2][3], [3][3], [3][4]. Κάθε κελί γεμίζεται ξεχωριστά:

fillCell(2, 2, value);

fillCell(2, 3, value);

fillCell(3, 3, value);

fillCell(3, 4, value);

Παράδειγμα 1 (LINE mode):

Ο χρήστης ξεκινά στο [4][1] και αφήνει το ποντίκι στο [4][5]. Το σύστημα αναγνωρίζει οριζόντια κίνηση και γεμίζει μαζικά:

fillCell(4, 1, value);

fillCell(4, 2, value);

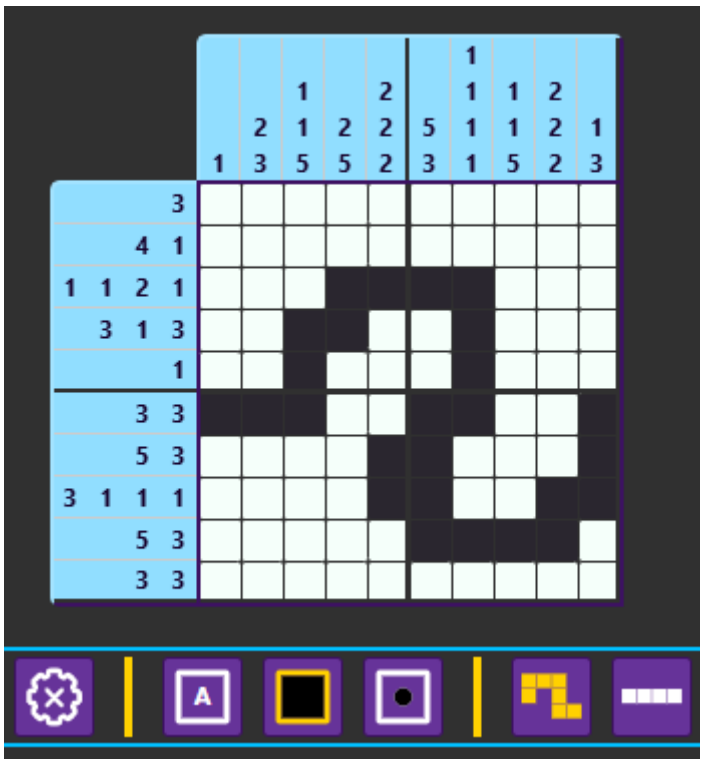
fillCell(4, 3, value);

fillCell(4, 4, value);

fillCell(4, 5, value);

Το ίδιο ισχύει για κάθετη κίνηση (σταθερό col, μεταβαλλόμενο row).

Αν η κίνηση δεν είναι ευθυγραμμισμένη (διαγώνια), η handleLineDrawing() την αγνοεί. Αυτό αποτρέπει μη έγκυρες συμπεριφορές και διατηρεί τη λογική των Nonogram, που δεν περιλαμβάνουν διαγώνιες γραμμές.



Εικόνα 3.20 Στιγμιότυπο από την εφαρμογή. Παράδειγμα συμπλήρωσης κελιών με FREE drawMode.

Το drawMode μπορεί να αλλάζει δυναμικά από το χρήστη μέσα από πλήκτρα του Toolbar, που ενεργοποιούν το mode LINE ή FREE (βλ. Εικόνα 3.19). Το ενεργό mode μεταφέρεται ως prop στο NonogramTable, όπου και καθορίζει τον τρόπο με τον οποίο θα συμπεριφερθούν οι onMouseEnter και handleLineDrawing.

Εικόνα 3.21 Στιγμιότυπο από την εφαρμογή. FREE σχεδίαση (αριστερό κουμπί) και LINE σχεδίαση (δεξί κουμπί).

Ο διαχωρισμός ανάμεσα σε drawMode και drawState είναι σημαντικός:

- drawState: τι σχεδιάζεται (π.χ. γεμισμένο, dot, auto).
- drawMode: πώς σχεδιάζεται (π.χ. ελεύθερα ή σε γραμμή).

Η ανεξαρτησία τους επιτρέπει συνδυασμούς (π.χ. dot σε γραμμή).

3.11.7 Υποστηρικτικές Συναρτήσεις και Structured Ενημερώσεις

Καθ' όλη τη διάρκεια της υλοποίησης της εφαρμογής Nonogram, κρίθηκε απαραίτητο να αναπτυχθούν και να χρησιμοποιηθούν επιμέρους βοηθητικές λειτουργίες και σταθερές που δεν ανήκουν άμεσα σε κάποιο από τα βασικά λειτουργικά modes της εφαρμογής, αλλά εξυπηρετούν θεμελιώδεις ανάγκες σε επίπεδο αξιοπιστίας, αποδοτικότητας και ευκολίας συντήρησης.

Οι συναρτήσεις αυτές ενισχύουν τη σταθερότητα του grid κατά τη σχεδίαση, εξασφαλίζουν ότι η διαχείριση της κατάστασης (state) είναι σύμφωνη με τις αρχές του React, επιτρέπουν καθαρή υλοποίηση undo/redo, και αποτρέπουν την εμφάνιση ανεπιθύμητων παρενεργειών (side effects) λόγω λανθασμένων μεταβολών σε κοινόχρηστα δεδομένα.

Η κατηγοριοποίηση και περιγραφή αυτών των μηχανισμών στην παρούσα ενότητα έχει ως στόχο:

- την τεκμηρίωση των υποδομών πάνω στις οποίες στηρίζονται βασικές λειτουργίες,
- την ανάδειξη καλών πρακτικών στην επεξεργασία πλεγμάτων,
- και την κατανόηση του τρόπου με τον οποίο εξασφαλίζεται η συνοχή των δεδομένων σε πραγματικό χρόνο.

Ακολουθούν οι σημαντικότεροι μηχανισμοί και δομικές στρατηγικές που ενσωματώθηκαν για την επίτευξη των παραπάνω στόχων.

Αντιγραφή Πινάκων – Shallow και Deep Copy

Η ανάγκη για δημιουργία νέων δομών δεδομένων σε κάθε τροποποίηση ενός πλέγματος αποτελεί σημαντική για την αποδοτική και αξιόπιστη διαχείριση κατάστασης σε περιβάλλοντα όπως η React. Στην παρούσα εφαρμογή, κάθε πλέγμα (grid) υλοποιείται ως πίνακας δισδιάστατου τύπου Array<Array<Cell>>, όπου κάθε Cell μπορεί να είναι είτε boolean είτε αντικείμενο.

Κατά τη μεταβολή ενός κελιού, δεν επιτρέπεται η απευθείας τροποποίηση του αρχικού grid (grid[row][col] = value) γιατί αυτό οδηγεί σε ανεπιθύμητα side-effects. Αντίθετα, δημιουργείται νέα δομή με shallow ή deep copy, ανάλογα με τη λειτουργία.

Παραδείγματα πρακτικής:

Σε fillCell, για ενημέρωση ενός μόνο κελιού:

```
const newRow = [...prev[row]];
newRow[col] = value;
const updated = [...prev];
updated[row] = newRow;
```

Σε περιπτώσεις ολικής ανανέωσης, όπως handleResetClick, δημιουργείται νέο grid με:

```
Array.from({ length: height }, () =>
  Array.from({ length: width }, () => ({ state: 0, x: false }))
);
```

Αυτές οι τεχνικές διασφαλίζουν ότι κάθε αλλαγή δημιουργεί νέο reference, επιτρέποντας στο React να ανιχνεύσει αλλαγές και να αποδώσει σωστά τη νέα κατάσταση. Αποφεύγονται σφάλματα που σχετίζονται με μη ορατές αλλαγές σε nested objects, και διατηρείται η αρχή της αμεταβλητότητας των props και του state.

Διαχείριση ιστορικού μέσω useUndoRedo

Η υλοποίηση ενός συστήματος Undo/Redo κρίνεται αναγκαία σε οποιαδήποτε εφαρμογή δημιουργικού ή λογικού περιεχομένου, και ειδικά σε περιβάλλοντα όπως το Nonogram όπου ο χρήστης πειραματίζεται, υποθέτει και επαληθεύει τη λύση του. Για τον σκοπό αυτό, η εφαρμογή αξιοποιεί custom React hook με το όνομα useUndoRedo, το οποίο αποθηκεύει τις προηγούμενες καταστάσεις του grid και επιτρέπει την ασφαλή επιστροφή ή επαναφορά ενεργειών.

Το hook αυτό περιλαμβάνει δύο στοίβες κατάστασης:

- undoStack: αποθηκεύει τις προηγούμενες καταστάσεις πλέγματος (history),
- redoStack: κρατά τις επόμενες καταστάσεις σε περίπτωση αναίρεσης που ο χρήστης επαναφέρει.

Η συνάρτηση setPlayingCellsWithHistory(newState) καλείται αντί για απλή setPlayingCells, και είναι υπεύθυνη για την ενημέρωση της τρέχουσας κατάστασης αλλά και για την καταγραφή της προηγούμενης στο undoStack. Αν η παράμετρος newLevel είναι true, τότε γίνεται reset του ιστορικού (όπως κατά τη φόρτωση νέου puzzle).

Προκειμένου να αποφευχθεί υπερφόρτωση μνήμης, εφαρμόζεται μέγιστο όριο ιστορικού μέσω της limitHistory(stack), το οποίο κόβει την στοίβα στο MAX_HISTORY = 50 βήματα. Αυτό εξασφαλίζει επαρκή βάθος ενεργειών αλλά και προβλεψιμότητα στη χρήση πόρων.

Η αναίρεση (handleUndo) ανακτά την τελευταία εγγραφή από το undoStack, την εφαρμόζει ως playingCells και μεταφέρει την τρέχουσα κατάσταση στο redoStack. Η επαναφορά (handleRedo) κάνει το αντίστροφο. Με αυτόν τον τρόπο υποστηρίζεται πλοήγηση μπρος-πίσω χωρίς απώλεια πληροφορίας.

Παράδειγμα χρήσης (drag 3 κελιών → undo → redo):

1. Ο χρήστης γεμίζει [3][4], [3][5], [3][6]. Όλα καταγράφονται μαζί.
2. Καλεί handleUndo() → το grid επιστρέφει στην προηγούμενη μορφή.
3. Καλεί handleRedo() → η ενέργεια επανέρχεται.

Ο σχεδιασμός αυτός εξασφαλίζει την αναίρεση ολόκληρων ενεργειών (όχι μεμονωμένων κελιών) καθώς και την ισορροπημένη χρήση μνήμης. Ακόμη παρέχει έλεγχο στις μεταβολές του πλέγματος και ασφάλεια κατά την χρήση των clues και των hints.

Διαχείριση Χρήση Σταθερών – CELL_STATE, DRAW_STATE, DRAW_MODE

Στο πλαίσιο της σωστής οργάνωσης του κώδικα και της αποφυγής σφαλμάτων, η εφαρμογή Nonogram βασίζεται στη χρήση προτυποποιημένων σταθερών (constants) για την αποτύπωση των καταστάσεων κελιών, εργαλείων σχεδίασης και λειτουργικών modes. Οι σταθερές αυτές βελτιώνουν την αναγνωσιμότητα, την επεκτασιμότητα και την αποσφαλμάτωση του λογισμικού.

Αντί να χρησιμοποιούνται απλοί αριθμοί (magic numbers) που δεν είναι αυτοεξηγούμενοι, η χρήση σταθερών καθιστά σαφές το νόημα κάθε τιμής. Επιπλέον, επιτρέπει τη συγκεντρωτική αλλαγή ορισμών χωρίς να χρειάζεται επεξεργασία πολλών σημείων του κώδικα.

CELL_STATE

Η σταθερά CELL_STATE ορίζει τις δυνατές καταστάσεις ενός κελιού κατά το Play Mode και χρησιμοποιείται εκτεταμένα για την αποθήκευση και την εμφάνιση της κατάστασης κάθε κελιού:

const CELL_STATE = { EMPTY: 0, FILLED: 1, DOT: 2};

Η τιμή EMPTY σημαίνει κενό κελί, FILLED σημαίνει γεμισμένο, και DOT δηλώνει κελί που έχει σημειωθεί ως ενδεχομένως γεμισμένο. Οι τιμές αποθηκεύονται στο πεδίο state κάθε κελιού.

DRAW_STATE

Η σταθερά DRAW_STATE χρησιμοποιείται για τον καθορισμό της συμπεριφοράς κατά τη σχεδίαση. Καθορίζει τι τύπου ενέργεια εκτελεί ο χρήστης όταν κάνει κλικ ή drag:

const DRAW_STATE = {AUTO: 0, FILL: 1, DOT: 2};

Η τιμή AUTO εφαρμόζει κυκλικά τις διαθέσιμες καταστάσεις (EMPTY → FILLED → DOT → EMPTY), ενώ οι FILL και DOT εισάγουν απευθείας την αντίστοιχη τιμή.

DRAW_MODE

Η σταθερά DRAW_MODE καθορίζει το αν η σχεδίαση γίνεται ελεύθερα (FREE) ή με ευθυγράμμιση (LINE). Χρησιμοποιείται σε συνδυασμό με το toolbar επιλογής και στο handleMouseEnter():

const DRAW_MODE = {FREE: 0, LINE: 1};

Σε LINE mode, η σχεδίαση εκτείνεται σε ευθεία (row ή col) από την αρχική θέση (drawStart) έως την τρέχουσα (row, col), ενώ στο FREE γεμίζεται μόνο το κελί πάνω από το οποίο περνά το ποντίκι.

Η χρήση σταθερών σε κάθε ένα από τα παραπάνω context έχει πρακτικά αποτελέσματα στη συντηρησιμότητα του κώδικα, μειώνει τον κίνδυνο λανθασμένων συγκρίσεων (state === 1 vs state === CELL_STATE.FILLED) και επιτρέπει στο IDE να προτείνει και να αυτοσυμπληρώνει τιμές με μεγαλύτερη ακρίβεια.

Υποστηρικτικά Helpers – Sets, sessionCellCoords, getDrawValue

Πέρα από τις βασικές λειτουργικές και δομικές μονάδες, η εφαρμογή Nonogram περιλαμβάνει και ορισμένα βοηθητικά στοιχεία που υποστηρίζουν κρίσιμες εσωτερικές διαδικασίες. Αυτά δεν αφορούν άμεσα το UI ή την κατάσταση solving, αλλά ενισχύουν την αξιοπιστία της ροής, την αποδοτικότητα και την απλότητα του κώδικα.

sessionCellCoords – Παρακολούθηση Κελιών σε Drag

Το sessionCellCoords είναι μια μεταβλητή τύπου Set που καταγράφει ποια κελιά έχουν επηρεαστεί κατά τη διάρκεια ενός drag session. Το κάθε στοιχείο είναι ένα string τύπου "row-col" (π.χ. "3-5"). Το Set επιλέχθηκε για αποτροπή διπλών καταχωρήσεων και γρήγορη αναζήτηση (.has(...)).

Χρησιμοποιείται για να εφαρμόζονται class names σε όσα κελιά έχουν «δραστηριοποιηθεί» κατά το τρέχον draw, βελτιώνοντας το visual feedback.

hintCells – Προσωρινή Επισήμανση Κελιών Hint

Το hintCells λειτουργεί επίσης ως Set, με ίδιο format ("row-col"). Περιέχει τα κελιά που έχουν τονιστεί (highlighted) ως υπόδειξη (Hint) στο πλέγμα. Χρησιμοποιείται αποκλειστικά από το NonogramTable, ώστε να αποδοθεί διαφορετικό CSS class σε όσα κελιά είναι ενεργά για λίγα δευτερόλεπτα μετά την handleHintClick().

getDrawValue() – Υπολογισμός Τιμής κατά το Drawing

Η getDrawValue() είναι inline helper συνάρτηση που καλείται μέσα στη handleMouseEnter. Ανάλογα με την drawAction, επιστρέφει είτε:

- { state: ..., x: false } για κανονική σχεδίαση, είτε
- { state: 0, x: true } για εισαγωγή X.

Με αυτόν τον τρόπο, συγκεντρώνεται η λογική απόδοσης τιμής σε μία μόνο helper, μειώνοντας την πολυπλοκότητα στο flow της σχεδίασης.

Ο συνδυασμός αυτών των helpers επιτρέπει στην εφαρμογή να διαχειρίζεται με σαφήνεια και συνέπεια την προσωρινή κατάσταση του πλέγματος κατά τη σχεδίαση, να εμφανίζει χρήσιμες επισημάνσεις (hint, session), και να διατηρεί το rendering γρήγορο και ευανάγνωστο.

3.12 Επίλογος

Σε αυτό το κεφάλαιο αναλύθηκε εκτενώς η τεχνική και λειτουργική υλοποίηση της εφαρμογής Nonogram, παρουσιάζοντας σε βάθος τη δομή του συστήματος, τα βασικά modes χρήσης και τον τρόπο με τον οποίο ο χρήστης αλληλεπιδρά με το περιβάλλον επίλυσης.

Αρχικά, περιγράφηκε η θεμελιώδης αρχιτεκτονική του πλέγματος, τόσο σε επίπεδο δεδομένων όσο και εμφάνισης. Παρουσιάστηκε η διαφορετική εσωτερική αναπαράσταση των κελιών ανάλογα με το ενεργό mode (Draw ή Play), καθώς και ο τρόπος παραγωγής των αριθμητικών ενδείξεων (clues) που καθοδηγούν τον παίκτη. Δόθηκε έμφαση στον μηχανισμό δημιουργίας και αποθήκευσης επιπέδων, αλλά και στη διαχείριση της λογικής κατάστασης μέσω hooks και state variables.

Έπειτα, έγινε αναλυτική παρουσίαση των βασικών λειτουργιών που προσφέρονται στο Play Mode, όπως η συμπλήρωση κελιών μέσω ποντικιού, η σήμανση πιθανών κελιών με X, η αυτόματη αναγνώριση ολοκληρωμένων clues, καθώς και η δυνατότητα αναίρεσης και επαναφοράς ενεργειών. Ειδική μνεία δόθηκε στη λειτουργία Preview, η οποία προσφέρει οπτική επιβεβαίωση της σωστής εικόνας, χωρίς να επηρεάζεται η διαδικασία επίλυσης. Ιδιαίτερο βάρος δόθηκε στον μηχανισμό υποδείξεων (hint system), ο οποίος αναλύθηκε τόσο σε επίπεδο λογικής όσο και σε επίπεδο κώδικα. Τεκμηριώθηκε η διαδικασία επιλογής των segment που προτείνονται ως υπόδειξη, με βάση την πρόσφατη δραστηριότητα του χρήστη, και παρουσιάστηκαν παραδείγματα εφαρμογής τους στο περιβάλλον χρήσης.

Επίσης, παρουσιάστηκαν ειδικές λειτουργίες υποστήριξης, όπως ο έλεγχος λύσης, η εφαρμογή αυτόματων X σε πλήρως επιλυμένες γραμμές/στήλες, και η χρήση της βιβλιοθήκης ReactCardFlip για την υλοποίηση δυναμικής εναλλαγής μεταξύ των views. Στην ίδια κατεύθυνση, έγινε ανάλυση του τρόπου με τον οποίο αξιοποιείται η IndexedDB (μέσω Dexie.js) για αποθήκευση επιπέδων, καθώς και της σημασίας της hook useLiveQuery για δυναμική παρακολούθηση αλλαγών στη βάση.

Τέλος, το κεφάλαιο ολοκληρώθηκε με την ενότητα των υποστηρικτικών μηχανισμών, όπου τεκμηριώθηκαν κρίσιμα τεχνικά σημεία: structured cloning για αποφυγή mutation, διαχείριση ιστορικού ενεργειών (undo/redo), χρήση σταθερών για αυξημένη αναγνωσιμότητα, και υποστηρικτικά helpers όπως τα sessionCellCoords, hintCells και getDrawValue.

Συνολικά, το κεφάλαιο αυτό ανέδειξε πώς οι σχεδιαστικές και προγραμματιστικές αποφάσεις υποστηρίζουν την εργονομία, την ακρίβεια και την ευελιξία της εφαρμογής. Η επόμενη ενότητα επικεντρώνεται πλέον στην αξιολόγηση της εφαρμογής από την πλευρά του χρήστη, την αποδοτικότητα των επιμέρους λειτουργιών και τις δυνατότητες μελλοντικής επέκτασης.

Κεφάλαιο 4ο: Αξιολόγηση, Βελτιώσεις και Προοπτικές

4.1 Εισαγωγή

Το παρόν κεφάλαιο επικεντρώνεται στην αξιολόγηση της λειτουργικής και τεχνικής πληρότητας της εφαρμογής, όπως αυτή υλοποιήθηκε στο πλαίσιο της παρούσας πτυχιακής εργασίας. Σκοπός της ενότητας είναι να εντοπίσει τα δυνατά και αδύνατα σημεία του λογισμικού, να επιβεβαιώσει την κάλυψη των απαιτήσεων που τέθηκαν στο στάδιο σχεδιασμού, και να καταγράψει πιθανές προτάσεις για βελτίωση ή μελλοντική επέκταση.

Η αξιολόγηση βασίζεται σε μια σειρά ποιοτικών και ποσοτικών κριτηρίων, όπως η πληρότητα λειτουργικότητας, η εμπειρία χρήστη, η απόδοση σε διαφορετικά περιβάλλοντα χρήσης, καθώς και η ευκολία συντήρησης και επεκτασιμότητας του κώδικα. Επιπλέον, εξετάζονται πρακτικά σενάρια χρήσης και λειτουργικής ροής, τα οποία εφαρμόζονται στην ίδια την εφαρμογή με σκοπό την επιβεβαίωση της ορθότητας και της σταθερότητάς της.

Η ενότητα αυτή δεν περιορίζεται σε έλεγχο λειτουργιών, αλλά επιχειρεί και μια ευρύτερη τεχνική αποτίμηση: κατά πόσο η αρχιτεκτονική επιλογή (ReactJS + Dexie.js + IndexedDB) υποστήριξε επιτυχώς τους στόχους της εφαρμογής, και ποιοι είναι οι τομείς στους οποίους μπορεί να υπάρξει τεχνική βελτίωση ή ενίσχυση της εμπειρίας του χρήστη.

4.2 Κριτήρια Αξιολόγησης

Για να αξιολογηθεί αν η εφαρμογή ανταποκρίνεται στους στόχους που είχαν τεθεί κατά τον σχεδιασμό της, εξετάζεται από διάφορες πλευρές: λειτουργικότητα, απόδοση, εμπειρία χρήστη και ποιότητα υλοποίησης. Με βάση αυτά, διαμορφώνονται συγκεκριμένα κριτήρια αξιολόγησης που βοηθούν να φανεί αν το τελικό αποτέλεσμα είναι λειτουργικό, χρήσιμο και σταθερό.

Αρχικά, βασικό κριτήριο είναι η πληρότητα των λειτουργιών. Εξετάζεται αν η εφαρμογή επιτρέπει στον χρήστη να δημιουργήσει, να αποθηκεύσει και να επιλύσει γρίφους όπως προβλέπεται. Περιλαμβάνονται λειτουργίες όπως η διαχείριση clues, τα εργαλεία συμπλήρωσης κελιών, η δυνατότητα προεπισκόπησης της λύσης, οι υποδείξεις (hints), η αυτόματη συμπλήρωση X και φυσικά η αναίρεση και επαναφορά ενεργειών. Στόχος είναι να διαπιστωθεί αν η εφαρμογή καλύπτει πλήρως όσα υπόσχεται.

Ακολουθεί η αξιολόγηση της εμπειρίας χρήστη (User Experience). Εξετάζεται το πόσο εύκολη και κατανοητή είναι η χρήση της εφαρμογής. Αυτό περιλαμβάνει τη διάταξη της διεπαφής, την οργάνωση των εργαλείων, την καθαρότητα των modes (Play, Edit, Preview), αλλά και το κατά πόσο ο χρήστης λαμβάνει οπτική ανατροφοδότηση για τις ενέργειές του. Για παράδειγμα, η σωστή σήμανση των clues ως ολοκληρωμένων ή η εμφάνιση X όπου χρειάζεται, βοηθούν σημαντικά στην επίλυση και ανταποκρίνονται σε βασικές αρχές σχεδίασης παιχνιδιών ως προς την καθοδήγηση του παίκτη και την αίσθηση προόδου [7].

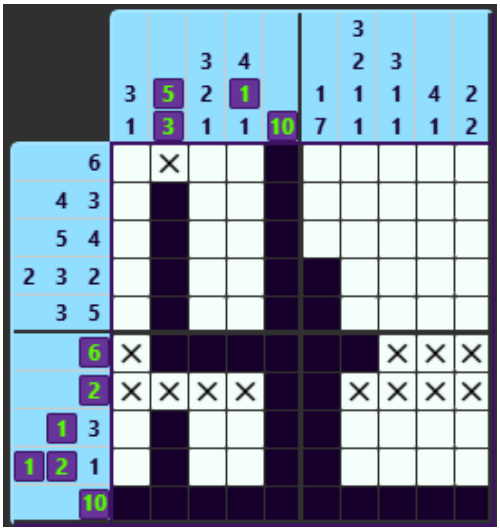
Τέλος, εξετάζεται η ποιότητα του ίδιου του κώδικα και αν αυτός έχει υλοποιηθεί με τρόπο που να επιτρέπει εύκολες μελλοντικές βελτιώσεις ή προσθήκες. Αξιολογείται αν η λογική είναι διαχωρισμένη σε modules και components, αν υπάρχουν επαναχρησιμοποιήσιμα κομμάτια όπως hooks και helpers, και αν η δομή του project είναι καθαρή και συντηρήσιμη.

Η συνολική αυτή αξιολόγηση παρέχει το υπόβαθρο για την ανάλυση που ακολουθεί με πραγματικά σενάρια χρήσης και συμπεράσματα για την κατάσταση της εφαρμογής.

4.3 Δοκιμή Εφαρμογής στην Πράξη

Η αξιολόγηση της εφαρμογής πραγματοποιήθηκε μέσω δοκιμών με στόχο να εξεταστεί η ομαλή λειτουργία όλων των βασικών εργαλείων της και η γενικότερη εμπειρία χρήστη. Η διαδικασία περιλάμβανε επαναλαμβανόμενη χρήση της εφαρμογής, τόσο σε σενάρια δημιουργίας νέου γρίφου, όσο και σε περιπτώσεις επίλυσης επιπέδων διαφορετικής δυσκολίας.

Κατά τη δημιουργία νέου επιπέδου μέσω του Edit Mode, ο χρήστης είχε τη δυνατότητα να ορίσει το μέγεθος του πλέγματος και να "ζωγραφίσει" τη λύση του γρίφου γεμίζοντας τα επιθυμητά κελιά. Οι αριθμητικές ενδείξεις (clues) υπολογίζονταν αυτόματα σε πραγματικό χρόνο, και η αποθήκευση του επιπέδου στη μνήμη του browser ολοκληρωνόταν με επιτυχία.



Εικόνα 4.1 Παράδειγμα επίλυσης γρίφου στο Play Mode, με χρήση Fill και X.

Κατά την επίλυση ενός γρίφου στο Play Mode, η αλληλεπίδραση με το πλέγμα ήταν άμεση και ομαλή. Η χρήση του εργαλείου Fill επέτρεπε τη συμπλήρωση κελιών είτε μεμονωμένα είτε μέσω drag. Επιπλέον, τα X προσθέτονταν χωρίς καθυστέρηση. Το grid ανταποκρινόταν σταθερά, χωρίς ανεπιθύμητα refresh ή καθυστερήσεις.

Ο μηχανισμός Undo/Redo αποδείχθηκε πολύ χρήσιμος κατά την επίλυση, καθώς ο χρήστης μπορούσε να αναιρέσει λάθη και να επιστρέψει σε προηγούμενες καταστάσεις του πλέγματος. Η λειτουργία καταγράφει ολόκληρες "κινήσεις", ειδικά κατά τη χρήση drag, και όχι μεμονωμένα clicks, γεγονός που κάνει την εμπειρία πολύ πιο φιλική.

Ο μηχανισμός Hint λειτουργούσε με συνέπεια, προτείνοντας χρήσιμες περιοχές για γεμίσματα με βάση το σημείο που ο χρήστης είχε πρόσφατα αλληλεπιδράσει. Οι υποδείξεις εμφανίζονταν με διαφορετικό χρώμα και βοηθούσαν στον εντοπισμό κελιών που όντως έπρεπε να συμπληρωθούν. Ο αριθμός διαθέσιμων hints μειωνόταν σωστά κάθε φορά που καλούνταν η λειτουργία.

Ο οπτικός έλεγχος των clues (μέσω autoMarkClues) και η αυτόματη συμπλήρωση X σε πλήρως επιλυμένες γραμμές ή στήλες (μέσω autoX) προσέφεραν σημαντική βοήθεια, ειδικά σε μεγαλύτερα επίπεδα. Οι λειτουργίες αυτές ενεργοποιούνταν χωρίς καθυστέρηση και βελτίωναν τη σαφήνεια της προόδου του παίκτη.

Η δυνατότητα Preview επέτρεπε στον χρήστη να δει την τελική εικόνα του γρίφου ανά πάσα στιγμή, χωρίς να διακόπτεται η πρόοδός του. Η εναλλαγή με το Play Mode γινόταν με ομαλή μετάβαση, επιτρέποντας στον δημιουργό ενός γρίφου να επιβεβαιώσει το οπτικό αποτέλεσμα πριν αποθηκεύσει.

Δοκιμάστηκε επίσης η λειτουργία εισαγωγής επιπέδων από αρχείο .xml, μέσω του περιβάλλοντος Edit Mode. Η επιλογή αρχείου οδήγησε σε σωστή απόδοση του grid και του τίτλου, με αυτόματη δημιουργία του πλέγματος και δυναμική παραγωγή των clues. Η αλληλεπίδραση συνέχισε κανονικά, όπως σε κάθε χειροκίνητα σχεδιασμένο επίπεδο, επιτρέποντας αποθήκευση και μετάβαση στο Play Mode. Η λειτουργία κρίθηκε σταθερή και ενσωματωμένη ομαλά στη συνολική εμπειρία χρήστη.

Συνολικά, όλες οι δοκιμές κατέληξαν στο συμπέρασμα ότι η εφαρμογή είναι σταθερή, πλήρως λειτουργική και αποδοτική στις περισσότερες περιπτώσεις χρήσης. Τα βασικά εργαλεία ανταποκρίνονται σωστά, η αποθήκευση λειτουργεί χωρίς απώλειες και η εμπειρία χρήσης είναι θετική.

4.4 Προβλήματα και Περιορισμοί

Κατά τη διάρκεια της ανάπτυξης της εφαρμογής εντοπίστηκαν ορισμένα σημεία που λειτουργούν ως περιορισμοί σε σχέση με την πληρότητα των δυνατοτήτων. Αν και η εφαρμογή επιτελεί με επιτυχία τον βασικό της σκοπό — τη δημιουργία και επίλυση γρίφων Nonogram — εξακολουθούν να υπάρχουν λειτουργίες που είτε δεν έχουν υλοποιηθεί είτε αποτελούν πεδίο μελλοντικής ενίσχυσης.

Ένας βασικός περιορισμός αφορά την αποκλειστική υποστήριξη ασπρόμαυρων (δηλαδή δυαδικών) πλεγμάτων. Η εφαρμογή αυτή τη στιγμή δεν υποστηρίζει έγχρωμους Nonograms, γεγονός που περιορίζει τη δημιουργική ελευθερία των χρηστών και τη δυνατότητα αποτύπωσης πιο σύνθετων σχεδίων. Η προσθήκη υποστήριξης για χρώμα απαιτεί ουσιαστικές αλλαγές τόσο στη δομή των κελιών όσο και στον τρόπο απεικόνισης των clues και των εργαλείων.

Επιπλέον, δεν υπάρχει έλεγχος μοναδικότητας λύσης κατά τη δημιουργία νέου επιπέδου. Ο χρήστης μπορεί να σχεδιάσει έναν γρίφο και να τον αποθηκεύσει, είναι αδύνατο να λυθεί με λογική. Αν και η εφαρμογή παράγει αυτόματα τις αριθμητικές ενδείξεις (clues), δεν επαληθεύει αν αυτές οδηγούν με βεβαιότητα σε μία και μοναδική λύση. Αυτό περιορίζει τη δυνατότητα χρήσης της ως εργαλείο για αυστηρά λογικά puzzles ή ως εκπαιδευτική πλατφόρμα.

Σε επίπεδο αποθήκευσης, η εφαρμογή λειτουργεί αποκλειστικά τοπικά, μέσω της IndexedDB του browser. Αν και αυτό προσφέρει πλήρη offline λειτουργικότητα, δεν υπάρχει δυνατότητα συγχρονισμού δεδομένων μεταξύ συσκευών ή δημιουργίας backup. Επομένως, τα επίπεδα που δημιουργούνται παραμένουν προσβάσιμα μόνο από τη συσκευή στην οποία αποθηκεύτηκαν.

Οι παραπάνω περιορισμοί δεν αναιρούν τη βασική λειτουργικότητα της εφαρμογής, ωστόσο αποτελούν σημεία με περιθώριο βελτίωσης. Στην επόμενη ενότητα παρουσιάζονται προτάσεις για την πιθανή επέκταση ή ενίσχυση αυτών των δυνατοτήτων στο μέλλον.

4.5 Προτάσεις Βελτίωσης και Επέκτασης

Παρότι η εφαρμογή καλύπτει πλήρως τη βασική εμπειρία δημιουργίας και επίλυσης Nonogram, υπάρχουν περιθώρια βελτίωσης και προσθήκης επιπλέον λειτουργιών που θα μπορούσαν να την εξελίξουν ακόμη περισσότερο. Ορισμένες από αυτές αφορούν την επέκταση του τρόπου σχεδίασης, άλλες σχετίζονται με δυνατότητες διαμοιρασμού και άλλες με την ποιότητα των παραγόμενων puzzles.

4.5.1 Έγχρωμα Nonogram

Η υποστήριξη έγχρωμων Nonogram θα επέτρεπε τη δημιουργία πιο σύνθετων και ευδιάκριτων σχεδίων. Σε αντίθεση με τα ασπρόμαυρα puzzles, όπου η πληροφορία περιορίζεται στην ύπαρξη ή μη κελιών, τα έγχρωμα puzzles επιτρέπουν την αποτύπωση πραγματικών εικόνων, αντικειμένων ή ακόμα και pixel art χαρακτήρων.

Για να υλοποιηθεί αυτή η δυνατότητα, απαιτείται τροποποίηση της δομής των κελιών ώστε να αποθηκεύεται και η πληροφορία χρώματος, π.χ. ως color: **#FF0000** ή ως index από έναν χρωματικό πίνακα (colorIndex: 0). Τα clues θα πρέπει επίσης να τροποποιηθούν, ώστε να ορίζουν όχι μόνο το πλήθος των συνεχόμενων γεμισμένων κελιών, αλλά και το χρώμα καθενός. Για παράδειγμα, μία γραμμή clues σε έγχρωμο puzzle μπορεί να είναι [{done: false, value: 3, color: (Κόκκινο Hex)}, { done: false, value: 1, color: (Μπλέ Hex) }, { done: true, value: 2, color: (Πράσινο Hex)].

Η διεπαφή χρήστη θα πρέπει να εμπλουτιστεί με εργαλεία επιλογής χρώματος, όπως ένα color picker ή ένα toolbar με προκαθορισμένα χρώματα. Κάθε χρώμα θα πρέπει να αποδίδεται και στο clue display, ώστε ο παίκτης να γνωρίζει τι ακριβώς καλείται να συμπληρώσει. Το preview mode θα πρέπει επίσης να αποτυπώνει την εικόνα με τα αντίστοιχα χρώματα, ώστε να επιβεβαιώνεται το αποτέλεσμα. Η λειτουργικότητα αυτή συναντάται σε πολλές σύγχρονες πλατφόρμες Nonogram, όπως η **Nonograms.org**, όπου υποστηρίζονται puzzles πολλών χρωμάτων.

Ενδεικτικά, μία τροποποιημένη εγγραφή κελιού θα μπορούσε να έχει τη μορφή:

{ state: 1, color: "#303030, x: false} ή με χρήση index: **{ state: 1, colorIndex: 2 }**

Η προσθήκη αυτής της συνατότητας θα απαιτούσε αλλαγές στη λογική σχεδίασης, στο validation clues system, καθώς και στο σύστημα αποθήκευσης (π.χ. αλλαγή στο schema της IndexedDB). Ωστόσο, αποτελεί μια από τις πλέον ουσιαστικές επεκτάσεις για χρήστες που ενδιαφέρονται για πιο δημιουργικά και καλλιτεχνικά puzzles.

4.5.2 Έλεγχο μοναδικότητας λύσης

Η δυνατότητα ελέγχου της ύπαρξης και μοναδικότητας λύσης κατά την αποθήκευση ενός νέου επιπέδου θα πρόσθετε σημαντική αξία στην εφαρμογή, καθώς θα εξασφάλιζε ότι ο γρίφος που δημιουργείται μπορεί όντως να λυθεί με λογική, και ότι οδηγεί σε μία και μόνο σωστή λύση. Αυτό είναι ιδιαίτερα σημαντικό σε περιβάλλοντα όπου ο χρήστης μοιράζεται τα puzzles με άλλους ή όταν τα χρησιμοποιεί για εκπαιδευτικούς σκοπούς.

Η υλοποίηση ενός τέτοιου ελέγχου μπορεί να βασιστεί είτε σε αναδρομικό αλγόριθμο εξαντλητικής αναζήτησης (backtracking), είτε μέσω διατύπωσης του προβλήματος ως CSP (Constraint Satisfaction Problem), όπου κάθε γραμμή και κάθε στήλη μετατρέπεται σε σύνολο περιορισμών βασισμένων στα clues. Το σύστημα αναζητά έγκυρες λύσεις που ικανοποιούν όλους τους περιορισμούς, και μπορεί να διακοπεί μόλις εντοπίσει δεύτερη λύση (π.χ. foundSolutions >= 2), ώστε να διαπιστωθεί αν η λύση είναι μοναδική ή όχι. Για παράδειγμα, κατά την αποθήκευση ενός νέου puzzle, η εφαρμογή θα μπορούσε να εκτελέσει έναν solver στο παρασκήνιο, ο οποίος προσπαθεί να λύσει το grid βάσει των clues. Αν εντοπίσει ότι δεν υπάρχει λύση, εμφανίζει προειδοποίηση τύπου:

"Το επίπεδο δεν είναι επιλύσιμο με βάση τις ενδείξεις."

Εάν εντοπίσει πολλαπλές λύσεις, προτείνει στο χρήστη να το τροποποιήσει ή να αποθηκευτεί με ένδειξη "περίπλοκο".

Μία προσέγγιση βασίζεται στη μοντελοποίηση του grid ως πρόβλημα περιορισμών (Constraint Satisfaction Problem), όπου κάθε σειρά και στήλη αντιστοιχεί σε μεταβλητές με πεδία τιμών που ικανοποιούν τις clues. Οι σχέσεις μεταξύ των μεταβλητών διαμορφώνουν ένα δίκτυο περιορισμών, το οποίο μπορεί να επιλυθεί με τεχνικές όπως forward checking και arc consistency. Εναλλακτικά, το πρόβλημα μπορεί να μετατραπεί σε λογικό σύστημα ικανοποίησης (SAT) και να επιλυθεί με αντίστοιχους αλγόριθμους. Οι μέθοδοι αυτές επιτρέπουν όχι μόνο την εύρεση μίας λύσης, αλλά και την καταμέτρηση όλων των πιθανών λύσεων ώστε να διαπιστωθεί η μοναδικότητα [9].

4.5.3 Cloud αποθήκευση

Η τρέχουσα χρήση της IndexedDB εξασφαλίζει offline λειτουργία και γρήγορη απόκριση, αλλά περιορίζει τη διαθεσιμότητα των αποθηκευμένων επιπέδων στη συσκευή στην οποία δημιουργήθηκαν. Ο χρήστης δεν μπορεί να αποκτήσει πρόσβαση στα δεδομένα του από άλλη συσκευή ή να συγχρονίσει επίπεδα μεταξύ desktop και κινητού, ούτε να δημιουργήσει αντίγραφα ασφαλείας σε περίπτωση απώλειας δεδομένων.

Η προσθήκη υποστήριξης για cloud αποθήκευση, μέσω μιας υπηρεσίας backend όπως το **Firebase** (Google) ή το **Supabase** (ανοιχτού κώδικα), θα προσέφερε έναν ασφαλή και ελεγχόμενο τρόπο αποθήκευσης των επιπέδων σε απομακρυσμένο διακομιστή. Ο χρήστης θα μπορούσε να συνδέεται με λογαριασμό (μέσω email/password, Google OAuth ή άλλης μεθόδου), και η εφαρμογή θα αποθηκεύει και συγχρονίζει τα puzzles του σε πραγματικό χρόνο.

Η βασική ροή χρήσης θα μπορούσε να είναι η εξής:

1. Ο χρήστης συνδέεται στον λογαριασμό του.
2. Τα puzzles αποθηκεύονται στο cloud, είτε αυτόματα είτε με χειροκίνητο.
3. Από άλλη συσκευή, ο χρήστης συνδέεται και ανακτά το ίδιο προφίλ με όλα τα επίπεδα.
4. Τα δεδομένα εμφανίζονται ταξινομημένα, με δυνατότητα preview, κατηγοριοποίηση ή ακόμη και κοινή χρήση.

Η υποδομή θα επέτρεπε και πιο σύνθετα χαρακτηριστικά στο μέλλον, διαμοιρασμό επιπέδων με φίλους μέσω link, δημιουργία “συλλογών” puzzles (collections) ή ορισμό επιπέδων ως δημόσια/ιδιωτικά.

Τεχνικά, η υλοποίηση μπορεί να στηριχθεί είτε σε NoSQL βάσεις (π.χ. Firestore του Firebase) είτε σε SQL-based λύσεις (π.χ. Supabase Postgres), με απλό schema αποθήκευσης: κάθε χρήστης συσχετίζεται με ένα array ή πίνακα από levels, τα οποία περιέχουν JSON με clues, cells και metadata. Η δυνατότητα αυτή δεν αναιρεί την υπάρχουσα local-first αρχιτεκτονική, αλλά θα μπορούσε να λειτουργεί προαιρετικά ως “cloud mode”, διατηρώντας την offline εμπειρία για όσους δεν θέλουν λογαριασμό.

4.5.4 Προσθετες επεκτάσεις λειτουργικότητας

Ορισμένες πρόσθετες επεκτάσεις μπορούν να ενισχύσουν ακόμη περισσότερο την εμπειρία χρήσης και να προσδώσουν επιπλέον χαρακτήρα στην εφαρμογή. Αν και δεν είναι απαραίτητες για τη βασική λειτουργία, ανοίγουν δυνατότητες για ποικιλία gameplay, προσωποποίηση και βελτιωμένη προσβασιμότητα.

Η εισαγωγή χαρακτηριστικών δυσκολίας (π.χ. "Εύκολο", "Μέτριο", "Δύσκολο") για κάθε puzzle θα επέτρεπε την ομαδοποίηση επιπέδων με βάση το μέγεθος ή την πολυπλοκότητα clues. Οι χρήστες θα

μπορούσαν να επισημαίνουν τη δυσκολία του puzzle κατά τη δημιουργία, ή αυτή να υπολογίζεται αυτόματα βάσει προκαθορισμένων μετρικών (π.χ. πυκνότητα κελιών, συμμετρία, χρήση hints).

Η δυνατότητα χρονομέτρησης της επίλυσης θα μπορούσε να προστεθεί ως “Timed Mode”. Ο παίκτης θα βλέπει το χρόνο που χρειάζεται για να λύσει ένα puzzle, κάτι που μπορεί να δημιουργήσει μια αίσθηση πρόκλησης ή ανταγωνισμού. Τα στατιστικά χρόνου θα μπορούσαν να αποθηκεύονται για μελλοντική προβολή ή κατάταξη.

Μια ακόμη προαιρετική επέκταση είναι η προσθήκη κουμπιού "Generate Puzzle", που δημιουργεί αυτόματα ένα απλό, συμμετρικό puzzle προκαθορισμένου μεγέθους. Αν και η γεννήτρια δεν είναι εύκολο να διασφαλίζει πλήρη λογική μοναδικότητα της λύσης, θα μπορούσε να λειτουργήσει ως εργαλείο έμπνευσης ή γρήγορης δημιουργίας προτύπων για επεξεργασία.

Τέλος, υποστήριξη θεμάτων εμφάνισης (themes) ή λειτουργιών προσβασιμότητας (π.χ. high contrast, dark mode, μεγάλες ενδείξεις) θα βελτίωνε τη χρήση της εφαρμογής σε διαφορετικά περιβάλλοντα και για διαφορετικές ανάγκες. Οι ρυθμίσεις αυτές μπορούν να αποθηκεύονται τοπικά, ώστε κάθε χρήστης να διαμορφώνει την εμπειρία του σύμφωνα με τις προτιμήσεις του.

Οι παραπάνω επεκτάσεις, αν και δεν αποτελούν βασικές απαιτήσεις για τη λειτουργία της εφαρμογής, αναδεικνύουν την προοπτική μελλοντικής εξέλιξης προς μια πιο ευέλικτη, διαμοιράσιμη και δημιουργική πλατφόρμα. Η προσθήκη τεχνικών χαρακτηριστικών όπως η υποστήριξη έγχρωμων puzzles ή η αποθήκευση στο cloud, σε συνδυασμό με επιλογές που ενισχύουν την εμπειρία χρήστη, όπως τα themes ή τα challenges, μπορούν να μετατρέψουν την εφαρμογή από ένα λειτουργικό εργαλείο επίλυσης Nonogram σε μια πλήρη και προσαρμόσιμη πλατφόρμα δημιουργίας, κοινής χρήσης και αλληλεπίδρασης με γρίφους λογικής.

4.6 Επίλογος

Στο παρόν κεφάλαιο πραγματοποιήθηκε μια συνολική αποτίμηση της λειτουργικότητας, της εμπειρίας χρήστη και της τεχνικής αρτιότητας της εφαρμογής. Μέσα από συγκεκριμένα σενάρια χρήσης επιβεβαιώθηκε η σταθερότητα, η συνέπεια και η πληρότητα των βασικών εργαλείων, καθώς και η ομαλή ροή της διεπαφής σε πραγματικές συνθήκες.

Παράλληλα, αναγνωρίστηκαν σημεία με περιθώριο βελτίωσης, όπως η απουσία ελέγχου λύσης κατά την αποθήκευση, η έλλειψη υποστήριξης για έγχρωμους γρίφους ή η περιορισμένη διασυνδεσιμότητα. Οι περιορισμοί αυτοί δεν επηρεάζουν την κύρια εμπειρία, αλλά καθορίζουν τη δυναμική της εφαρμογής σε μελλοντικό πλαίσιο εξέλιξης.

Τέλος, διατυπώθηκαν τεκμηριωμένες προτάσεις επέκτασης, τόσο σε τεχνικό όσο και σε λειτουργικό επίπεδο, με στόχο τη μετατροπή της εφαρμογής σε ένα πιο ανοικτό, δημιουργικό και προσωποποιήσιμο εργαλείο. Το επόμενο κεφάλαιο επικεντρώνεται πλέον στη σύνθεση των συμπερασμάτων της εργασίας και στην επισήμανση των τελικών στόχων που επιτεύχθηκαν.

Κεφάλαιο 5ο: Συμπεράσματα

5.1 Επισκόπηση Στόχων και Επιτευγμάτων

Η εργασία είχε ως κύριο στόχο την ανάπτυξη μιας πλήρως λειτουργικής, web-based εφαρμογής για τη δημιουργία και επίλυση γρίφων Nonogram (Picross), με έμφαση στην εργονομία, την offline λειτουργία και την αυτονομία του χρήστη.

Πιο συγκεκριμένα, τέθηκαν οι εξής επιμέρους στόχοι:

- Ανάπτυξη περιβάλλοντος σχεδίασης (Draw Mode) για custom puzzles.
- Υλοποίηση πλήρους λειτουργικότητας επίλυσης (Play Mode), με υποστήριξη clues, εργαλείων, undo/redo, preview.
- Ενσωμάτωση μηχανισμών όπως αυτόματη σήμανση clues και σύστημα υποδείξεων.
- Αποθήκευση επιπέδων τοπικά μέσω IndexedDB, χωρίς εξάρτηση από backend.
- Εισαγωγή έτοιμων επιπέδων

Όλοι οι παραπάνω στόχοι υλοποιήθηκαν επιτυχώς, ενώ παράλληλα προστέθηκαν και χαρακτηριστικά που ενισχύουν την εμπειρία χρήστη, όπως responsive πλέγμα, προεπισκόπηση εικόνας και οπτική ανατροφοδότηση κατά τη διάρκεια επίλυσης. Συνολικά, η εφαρμογή ανταποκρίθηκε στις απαιτήσεις που τέθηκαν εξαρχής, και το τελικό αποτέλεσμα είναι πλήρως λειτουργικό και επεκτάσιμο. Τα συμπεράσματα της υλοποίησης αναλύονται στη συνέχεια.

5.2 Συμπεράσματα από την Υλοποίηση

Η εφαρμογή ανταποκρίθηκε με συνέπεια στους βασικούς άξονες αξιολόγησης: σταθερότητα λειτουργίας, πληρότητα εργαλείων, εμπειρία χρήστη και καθαρή αρχιτεκτονική κώδικα. Το περιβάλλον επίλυσης είναι γρήγορο, διαδραστικό και προσφέρει εργαλεία που υποστηρίζουν την εξελικτική λογική σκέψη, χωρίς να αφαιρούν από την πρόκληση του γρίφου.

Η ευχρηστία στη σχεδίαση και στην επίλυση είναι εμφανής στις τρεις λειτουργικές καταστάσεις της εφαρμογής (Edit, Play, Preview), οι οποίες επιτρέπουν στον χρήστη να μεταβαίνει με ευκολία και σαφήνεια μεταξύ δημιουργίας και επίλυσης. Ο οπτικός διαχωρισμός του πλέγματος, η δυναμική παραγωγή clues και η προαιρετική ενεργοποίηση εργαλείων όπως auto-X και hints ενισχύουν σημαντικά τη συνολική εμπειρία.

Η τεχνική υλοποίηση διακρίνεται για τη σταθερότητα και την απόδοση. Το grid αποδίδει σωστά ακόμη και σε μεσαία ή μεγάλα επίπεδα, ενώ οι λειτουργίες αναίρεσης, υποδείξεων και αυτόματων συμπληρώσεων εκτελούνται χωρίς καθυστέρηση.

Η αποθήκευση επιπέδων γίνεται τοπικά μέσω της IndexedDB με χρήση της βιβλιοθήκης Dexie.js, διασφαλίζοντας πλήρη offline λειτουργία, ανεξαρτησία από εξωτερικές υπηρεσίες και μονιμότητα αποθήκευσης.

Αξιοσημείωτη είναι και η καθαρή αρχιτεκτονική της εφαρμογής. Ο διαχωρισμός της λογικής σε components, hooks και βοηθητικά modules επιτρέπει την ευκολότερη επαναχρησιμοποίηση κώδικα, τη συντήρηση και τη μελλοντική επέκταση. Ενδεικτικά αποσπάσματα του κώδικα παρατίθενται στο Παράρτημα Α.

5.3 Προοπτικές και Τελικό Σχόλιο

Η εφαρμογή που αναπτύχθηκε καλύπτει τις βασικές ανάγκες δημιουργίας και επίλυσης γρίφων Nonogram, προσφέροντας ένα σύγχρονο, σταθερό και λειτουργικό περιβάλλον. Παρότι δεν εξαρτάται από εξωτερικές υπηρεσίες, η αρχιτεκτονική της επιτρέπει τεχνικά την ενσωμάτωση προηγμένων δυνατοτήτων, όπως έλεγχος λογικής μοναδικότητας, υποστήριξη έγχρωμων puzzles, ή διαμοιρασμός επιπέδων.

Ως πλατφόρμα, μπορεί να αξιοποιηθεί τόσο για ψυχαγωγική χρήση όσο και για εκπαιδευτικά ή γνωστικά παιχνίδια λογικής. Η εσωτερική δομή του κώδικα και ο τρόπος οργάνωσης των δεδομένων επιτρέπουν την περαιτέρω ανάπτυξή της χωρίς σημαντικές επεμβάσεις στον πυρήνα του συστήματος.

Το έργο αυτό αποτέλεσε μια ουσιαστική άσκηση εφαρμογής γνώσεων σε τεχνολογίες frontend, ενίσχυσε την κατανόηση του ReactJS οικοσυστήματος, και έδωσε την ευκαιρία να σχεδιαστεί μια εφαρμογή που συνδυάζει εμπειρία χρήστη, απόδοση και λογική σκέψη.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] T. Nishio and N. Ishida, *The World's Most Challenging Puzzles*, Tokyo: Nikoli Publishing, 1991.

[2] Nintendo Co., Ltd., *Mario's Picross* [Game], Game Boy, Japan: Nintendo, 1995.

[3] "Nonograms.org," [Online]. Available: https://www.nonograms.org. [Accessed: 10-May-2025].

[4] J. Wolter, *Web Paint-by-Number Puzzles (WebPBN)*. [Online]. Available: https://webpbn.com. [Accessed: 22-May-2025]

[5] K. Stolee and S. Elbaum, "Refining gameplay interaction with logic puzzles," *Empirical Software Engineering*, vol. 19, no. 4, pp. 918–955, 2014.

[6] Meta (Facebook Inc.), *React – A JavaScript library for building user interfaces*, [Online]. Available: https://reactjs.org

[7] M. Atwood, *Dexie.js Documentation*, [Online]. Available: https://dexie.org/docs.

[8] H. Desurvire, M. Caplan, and J. A. Toth, "Using heuristics to evaluate the playability of games," in *CHI '04 Extended Abstracts on Human Factors in Computing Systems*, Vienna, Austria: ACM, 2004, pp. 1425–1428.

[9] Mozilla Developer Network, "IndexedDB API", https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

[10] K. J. Batenburg and W. A. Kosters, "Constructing puzzles and puzzle solvers for Nonograms," *Information Sciences*, vol. 179, no. 10, pp. 1395–1407, 2009. DOI: 10.1016/j.ic.2008.10.004.

[11] Meta. *React Documentation*. [Online]. Available: https://react.dev

[12] Mozilla Developer Network. *IndexedDB API Documentation*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

ΠΑΡΑΡΤΗΜΑ Α : Αποσπάσματα Κώδικα

A.1 Συνάρτηση handleMouseDown (NonogramTable.js)

```
290  const handleMouseDown = (row, col, event) => {
291      event.preventDefault();
292      setIsDrawing( value: true);
293      props.onCellClick && props.onCellClick(row, col);
294      if (props.drawMode === DRAW_MODES.LINE) {
295          setDrawStart( value: { row, col }); // Set the coords of the starting point
296      }
297      const initialSession = props.cells.map(row => row.map(cell => isEditMode ? cell : { ...cell }));
298      setSessionCells(initialSession);
299
300      const current = props.cells[row][col];
301      if (isEditMode) {
302          const newVal = !current;
303          setFirstDrawValue(newVal);
304          fillCell(row, col, newVal);
305      } else {
306          const current = props.cells[row][col];
307          if (event.button === 2) {
308              // Right click: toggle X
309              const newVal = !current.x;
310              setDrawAction( value: { type: "x", value: newVal });
311              fillCell(row, col,  value: {state: CELL_STATES.EMPTY, x: newVal});
312          } else {
313              // Left click: determine target state to draw
314              let nextState;
315              if (props.drawState === drawStates.AUTO) {
316                  nextState = (current.state + 1) % Object.keys(CELL_STATES).length;
317              } else {
318                  const current = props.cells[row][col];
319                  if (props.drawState === drawStates.FILL) {
320                      nextState = current.state !== CELL_STATES.FILLED ? CELL_STATES.FILLED : CELL_STATES.EMPTY;
321                  } else {
322                      nextState = current.state !== CELL_STATES.DOT ? CELL_STATES.DOT : CELL_STATES.EMPTY;
323                  }
324              }
325              setDrawAction( value: { type: "state", value: nextState });
326              fillCell(row, col,  value: {state: nextState, x: false});
327          }
```

A.2 Συνάρτηση handleMouseEnter (NonogramTable.js)

```
const handleMouseEnter = (row, col) => {
    const getDrawValue = () => drawAction.type === "state" ?
        {state: drawAction.value, x: false} :
        {state: 0, x: false}

    const handleLineDrawing = () => {...}

    if (isDrawing && sessionCells) {
        if (isEditMode && firstDrawValue !== null) {
            fillCell(row, col, firstDrawValue);
        } else if (isPlayMode && drawAction) {
            if (props.drawMode === DRAW_MODES.LINE) {
                handleLineDrawing();
            } else {
                fillCell(row, col, getDrawValue());
            }

            props.onCellClick && props.onCellClick(row, col);
        }
    }
};
```

A.3 Συναρτήσεις handleUndo και handleRedo (Game.js)

```
const limitHistory = (stack) => stack.length > MAX_HISTORY ?
    stack.slice(stack.length - MAX_HISTORY) : stack; // Slice the array from end

const handleUndo = () => {
    setUndoStack( value: (prevUndo : ) => {
        let undoState = prevUndo;
        if (prevUndo.length) {
            const previousState = prevUndo[prevUndo.length - 1];
            setPlayingCells(previousState);
            setRedoStack( value: (prevRedo : ) => limitHistory( stack: [...prevRedo, playingCells]));
            undoState = prevUndo.slice(0, -1);
        }
        return undoState;
    });
};

const handleRedo = () => {
    setRedoStack( value: (prevRedo : ) => {
        let redoState = prevRedo;
        if (prevRedo.length) {
            const nextState = prevRedo[prevRedo.length - 1];
            setPlayingCells(nextState);

            setUndoStack( value: (prevUndo : ) => limitHistory( stack: [...prevUndo, playingCells]));
            redoState = prevRedo.slice(0, -1);
        }
        return redoState
    });
};
```

A.4 Απόσπασμα από την συνάρτηση handleHintClick (Game.js). Εμφανίζεται το σημείο όπου γίνεται ο εντοπισμός προτεραιότητας

```
if (segments.length) {
    if (lastFilledCell) {
        const relevantSegments = segments.filter(seg =>
            (seg.type === "row" && seg.index === lastFilledCell.row) ||
            (seg.type === "col" && seg.index === lastFilledCell.col)
        );
        if (relevantSegments.length) {
            prioritizedSegment = relevantSegments.reduce((a :T , b :T ) =>
                (b.end - b.start) > (a.end - a.start) ? b : a
            );
        }
    }

    if (!prioritizedSegment) {
        const incorrectSegments = segments.filter(hasIncorrectCells);
        if (incorrectSegments.length) {
            prioritizedSegment = incorrectSegments.reduce((a, b) =>
                (b.end - b.start) > (a.end - a.start) ? b : a
            );
        } else {
            // Fallback: Pick the longest available unfilled segment
            const maxLength = Math.max(...segments.map(s => s.end - s.start));
            const longestSegments = segments.filter(s => (s.end - s.start) === maxLength);
            prioritizedSegment = longestSegments[Math.floor( x: Math.random() * longestSegments.length)];
        }
    }
```

A.5 Απόσπαστμα από την συνάρτηση handleHintClick (Game.js). Εμφανίζεται το σημείο όπου δημιουργείται και ανανεώνεται ο newHintCells πίνακας και η ανανέωση των υπόλοιπων state.

```
if (prioritizedSegment) {
    const updated = playingCells.map(row => row.map(cell => ({...cell})));
    const newHintCells = [];

    if (prioritizedSegment.type === "row") {
        for (let col = prioritizedSegment.start; col <= prioritizedSegment.end; col++) {
            updated[prioritizedSegment.index][col] = {state: CELL_STATE.FILLED, x: false};
            newHintCells.push(`${prioritizedSegment.index}-${col}`);
        }
    } else {
        for (let row = prioritizedSegment.start; row <= prioritizedSegment.end; row++) {
            updated[row][prioritizedSegment.index] = {state: CELL_STATE.FILLED, x: false};
            newHintCells.push(`${row}-${prioritizedSegment.index}`);
        }
    }

    setPlayingCellsWithHistory(updated);
    setHintsCount( value: count => count - 1);
    setHintCells(new Set(newHintCells));
    setHintedClue( value: {type: prioritizedSegment.type, index: prioritizedSegment.index});

    // setTimeout(() => setHintCells(new Set()), 2000);
}
```

A.6 Συνάρτηση handleCheckSolvedClick (Game.js)

```
const handleCheckSolvedClick = () => {
    const isSolved = originalCells.every((row, rowIndex :number ) =>
        row.every((cell, colIndex) => {
            const original = cell.state === CELL_STATE.FILLED;
            const current = playingCells[rowIndex][colIndex].state === CELL_STATE.FILLED;
            return original === current;
        })
    );
    if (isSolved) {
        alert("🎉 Puzzle Solved!");

        setIsSolved( value: true);
        setHintCells(new Set()); // To stop the flashing
        setHintedClue( value: null); // To stop the flashing
    } else {
        alert("❌ Not solved yet. Keep going!");
    }
};
```

A.7 Συνάρτηση handleSaveLevelClick (Game.js)

```
    const handleSaveLevelClick = async () => {
        event.preventDefault();
        const isLevelNameValid = () => levelName.length && levelName.match( matcher: /^[\w\-\s]+$/);
        if (isLevelNameValid() && cells.some((row) => {
            console.log("row", row);
            return row.some((cell) => cell)
        })) {
            await dbLevels.add({
                name: levelName,
                cells: cells,
                width: cells[0].length,
                height: cells.length,
                solved: false
            });
        } else {
            setInvalidLevelName( value: true);
        }
    };
```

A.8 Συνάρτηση getLevelCells (Game.js)

```
const getLevelCells = async () => {
    const selectedLevelData = await dbLevels.get(selectedLevel);
    setCells(selectedLevelData.cells);
    setOriginalCells(
        selectedLevelData.cells.map(
            row => row.map(cell => typeof cell === "object" ?
                cell : cell ?
                    { state: CELL_STATE.FILLED, x: false } :
                    { state: CELL_STATE.EMPTY, x: false })));
}
```

A.9 Συνάρτηση fillClues (NonogramTable.js)

```
const fillClues = (grid) => {
    const getLineClues = (line) => {
        let clues = [];
        let count = 0;
        line.forEach((cell) => {
            const filled = typeof cell === "object" ? cell.state === CELL_STATES.FILLED : cell;
            if (filled) {
                count++;
            } else {
                if (count > 0) {
                    clues.push({done: false, value: count});
                    count = 0;
                }
            }
        });
        if (count > 0) clues.push({done: false, value: count});
        return clues.length ? clues : [];
    }

    let tempRowClues = grid.map(getLineClues);
    const transposedGrid = grid[0].map((_, i) => grid.map((row) => row[i]));
    let tempColClues = transposedGrid.map(getLineClues);

    setRowClues(tempRowClues);
    setColClues(tempColClues);
};
```

A.10 Συνάρτηση checkClueDone

```
const checkClueDone = (line, clues) => {
    let filledCount = 0;
    let clueIndex = 0;
    let matches = new Array(clues.length).fill( value: false);

    // Traverse the line and match filled cells to clues
    for (let i = 0; i < line.length; i++) {
        const cell = line[i];
        const filled = typeof cell === "object" ? cell.state === CELL_STATES.FILLED : cell === true;

        if (filled) {
            filledCount++;
        } else {
            if (filledCount > 0) {
                if (clueIndex < clues.length && clues[clueIndex].value === filledCount) {
                    matches[clueIndex] = true;
                }
                filledCount = 0;
                clueIndex++;
            }
        }
    }

    if (filledCount > 0 && clueIndex < clues.length && clues[clueIndex].value === filledCount) {
        matches[clueIndex] = true;
    }

    return matches;
};
```

A.11 Συνάρτηση fillCell

```
const fillCell = (row, col, value) => {
    setSessionCells( value: (prev) => {
        // let updated = prev.map((r) => [...r]); // Deep copy
        const newRow = [...prev[row]];
        newRow[col] = value;
        let updated = [...prev];
        updated[row] = newRow;

        let updatedClues;
        if (props.autoMarkClues) {
            updatedClues = markCluesAsDone(updated);
        }

        if (props.autoAddX && updatedClues) {
            updated = applyAutoX(updated, updatedClues.updatedRowClues, updatedClues.updatedColClues);
        }

        return updated;
    });
    setSessionCellCoords( value: (prevCoords) => new Set(prevCoords).add(`${row}-${col}`));
};
```

A.12 Συνάρτηση applyAutoX

```
const applyAutoX = (grid, rowCluesSet :any[]  = rowClues, colCluesSet :any[]  = colClues) => {
    const updated = grid.map(row => row.map(cell => ({ ...cell })));

    for (let row = 0; row < updated.length; row++) {
        const rowClues = rowCluesSet[row];
        const rowDone = rowClues?.every(clue => clue.done);

        if (rowDone) {
            updated[row] = updated[row].map(cell =>
                cell.state !== CELL_STATES.FILLED ? { state: CELL_STATES.EMPTY, x: true } : { ...cell }
            );
        }
    }

    for (let col = 0; col < updated[0].length; col++) {
        const colClues = colCluesSet[col];
        const colDone = colClues?.every(clue => clue.done);

        if (colDone) {
            for (let row = 0; row < updated.length; row++) {
                const cell = updated[row][col];
                if (cell.state !== CELL_STATES.FILLED) {
                    updated[row][col] = { state: CELL_STATES.EMPTY, x: true };
                }
            }
        }
    }

    return updated;
};
```

A.13 Συνάρτηση getUnfilledSegments

```
const getUnfilledSegments = (originalGrid, playingGrid) => {
    const segments = [];
    const height = originalGrid.length;
    const width = originalGrid[0].length;
    // Check rows
    for (let row = 0; row < height; row++) {
        let start = null;
        for (let col = 0; col <= width; col++) {
            const isFilled = col < width && originalGrid[row][col].state === CELL_STATE.FILLED;
            if (isFilled) {
                if (start === null) start = col;
            } else if (start !== null) {
                const end = col - 1;
                const alreadyFilled = playingGrid[row].slice(start, end + 1).every(cell => cell.state === CELL_STATE.FILLED);
                if (!alreadyFilled) {
                    segments.push({ type: 'row', index: row, start, end });
                }
                start = null;
            }}}
    // Check columns
    for (let col = 0; col < width; col++) {
        let start = null;
        for (let row = 0; row <= height; row++) {
            const isFilled = row < height && originalGrid[row][col].state === CELL_STATE.FILLED;
            if (isFilled) {
                if (start === null) start = row;
            } else if (start !== null) {
                const end = row - 1;
                const alreadyFilled = playingGrid.slice(start, end + 1).every(row => row[col].state === CELL_STATE.FILLED);
                if (!alreadyFilled) {
                    segments.push({ type: 'col', index: col, start, end });
                }
                start = null;
            }}}
    return segments;
};
```