Εργασία

Μαρία Σιώπη,Αφροδίτη Πιστικοπούλου,Μαρία Μιχαλέρη

Απρίλιος 2025

Περιεχόμενα

1 Εισαγωγή Motivation

1.1 Το θέμα με το οποίο ασχολείται η εργασία

Το παρόν project ασχολείται με την ανάπτυξη μιας εφαρμογής παιχνιδιού Sudoku χρησιμοποιώντας τη γλώσσα προγραμματισμού C++ και τις αρχές του αντικειμενοστραφούς προγραμματισμού (OOP). Ο χρήστης έχει τη δυνατότητα να παίξει Sudoku διαλέγοντας ανάμεσα από τρία επίπεδα δυσκολίας (Easy, Medium και Hard), να κερδίσει πόντους με κάθε σωστή απάντηση που βρίσκει, αν δυσκολευτεί να αγοράσει βοήθειες και σε περίπτωση που μείνει χωρίς ζωές (τρία επιτρεπτά λάθη) να αγοράσει άλλες τρείς αν έχει αρκετούς πόντους.

1.2 Ποιο πρόβλημα λύνει

Η εφαρμογή προσφέρει έναν διαδραστικό τρόπο εξάσκησης του νου μέσω ενός δημοφιλούς λογικού παιχνιδιού, καθώς και ένα παράδειγμα σωστής χρήσης των αρχών του OOP. Λύνει επίσης το πρόβλημα της δυσκολίας στην κατανόηση και οργάνωση ενός σύνθετου προβλήματος (όπως το Sudoku) μετατρέποντάς το σε ένα σύνολο από ευδιάκριτα αντικείμενα με ξεκάθαρες ευθύνες.

1.3 Λόγοι για την επιλογή του αντικείμενου

Επιλέξαμε να υλοποιήσουμε το παιχνίδι Sudoku γιατί μας φάνηκε μια ενδιαφέρουσα και προσιτή πρόκληση για να εφαρμόσουμε στην πράξη τις βασικές αρχές του αντικειμενοστραφούς προγραμματισμού. Το Sudoku είναι ένα παιχνίδι που συνδυάζει λογική και στρατηγική, κάτι που μας έδωσε τη δυνατότητα να σχεδιάσουμε και να οργανώσουμε τον κώδικα με σαφείς κλάσεις και λειτουργίες. Παράλληλα, θέλαμε να δημιουργήσουμε κάτι διαδραστικό και ευχάριστο στον χρήστη, και μέσα από αυτή τη διαδικασία καταφέραμε να κατανοήσουμε καλύτερα έννοιες όπως η κληρονομικότητα, η επαναχρησιμοποίηση κώδικα και ο διαχωρισμός λειτουργιών.

2 Objective & Scope

2.1 Βασικοί στόχοι

Ο βασικός στόχος της εργασίας ήταν η υλοποίηση ενός διαδραστικού παιχνιδιού Sudoku με χρήση αντικειμενοστραφούς προγραμματισμού στη γλώσσα C++. Μέσα από αυτή την υλοποίηση, επιδιώξαμε:

- Να κατανοήσουμε και να εφαρμόσουμε βασικές έννοιες του αντικειμενοστραφούς προγραμματισμού, όπως η δημιουργία και χρήση κλάσεων, η ενθυλάκωση (encapsulation) και η διαχωρισμένη ευθύνη ανάμεσα σε διαφορετικά μέρη του προγράμματος.
- Να ενισχύσουμε την εμπειρία του παίκτη ενσωματώνοντας στοιχεία όπως πόντους, σύστημα λαθών, βοήθεια και επίπεδα δυσκολίας.
- Να δομήσουμε τον κώδικα με τέτοιο τρόπο ώστε να είναι επεκτάσιμος, π.χ. μελλοντικά να μπορεί να υποστηρίξει γραφικό περιβάλλον (GUI).
- Να καλλιεργήσουμε δεξιότητες συνεργασίας και ομαδικής εργασίας, καθώς το έργο αυτό αναπτύχθηκε στο πλαίσιο ομαδικής προσπάθειας.

Ο στόχος μας δεν ήταν μόνο η λειτουργικότητα του Sudoku, αλλά και η εμπέδωση καλών προγραμματιστικών πρακτικών.

2.2 Δυνατότητες που περιλαμβάνονται

Η εφαρμογή Sudoku που υλοποιήσαμε περιλαμβάνει τις εξής δυνατότητες:

- Επιλογή επιπέδου δυσκολίας (εύκολο, μεσαίο, δύσκολο), το οποίο επηρεάζει τον αριθμό των προγεμισμένων κελιών.
- Δυναμική δημιουργία ενός έγκυρου Sudoku puzzle και ταυτόχρονη αποθήκευση της λύσης του.
- Διαδραστική είσοδος από τον χρήστη για τοποθέτηση αριθμών σε κενά κελιά.
- Έλεγχος ορθότητας κάθε κίνησης με αντίστοιχη ενημέρωση του συστήματος πόντων:
  - Κάθε σωστή απάντηση προσθέτει 2 πόντους.
  - Κάθε λανθασμένη απάντηση αφαιρεί 2 πόντους και αυξάνει τα λάθη.
- Δυνατότητα “αγοράς” βοήθειας για συμπλήρωση ενός κελιού, χρησιμοποιώντας πόντους.
- Δυνατότητα “αγοράς” επιπλέον προσπαθειών αν ο χρήστης εξαντλήσει τις 3 αρχικές.

- Εμφάνιση του χρόνου επίλυσης στο τέλος του παιχνιδιού.
- Εμφάνιση της τελικής λύσης ανεξαρτήτως έκβασης του παιχνιδιού.

Οι παραπάνω δυνατότητες συνδυάζουν τη λογική του παιχνιδιού με ένα στοιχείο στρατηγικής διαχείρισης πόντων και προσπαθειών, κάνοντάς το πιο ενδιαφέρον και απαιτητικό για τον παίκτη.

3 System Architecture

3.1 Τρόπος λειτουργίας του συστήματος

Η εφαρμογή που υλοποιήσαμε είναι ένα παιχνίδι Sudoku στο οποίο ο χρήστης μπορεί να επιλέξει επίπεδο δυσκολίας (εύκολο, μεσαίο, δύσκολο) και να παίξει σε έναν πίνακα 9x9 που δημιουργείται αυτόματα. Το παιχνίδι προσφέρει βαθμούς ανάλογα με τις σωστές απαντήσεις και αφαιρεί πόντους για λάθη, δίνοντας έτσι ένα επιπλέον κίνητρο στον παίκτη. Σε περίπτωση που ο παίκτης αντιμετωπίζει δυσκολία, μπορεί να ζητήσει βοήθεια, η οποία είναι δωρεάν. Αν κατά τη διάρκεια του παιχνιδιού ο παίκτης ξεμείνει από προσπάθειες και έχει πόντους μεγαλύτερους ή ίσους του πέντε τότε ο παίκτης μπορεί να αγοράσει τρεις νέες προσπάθειες. Ο παίκτης έχει περιορισμένες προσπάθειες και η διαδικασία ολοκληρώνεται είτε με επιτυχία (λύση του πίνακα) είτε με έξοδο από το παιχνίδι ή αποτυχία λόγω λαθών.

3.2 Βασικές συνιστώσες

Συνάρτηση `main()`
Η `main()` αποτελεί το σημείο εκκίνησης του προγράμματος. Περιλαμβάνει την αρχικοποίηση του πίνακα Sudoku, την επιλογή επιπέδου δυσκολίας από τον χρήστη, τη δημιουργία της λύσης και την εκκίνηση του παιχνιδιού μέσω της `playSudoku()`. Επίσης, καταγράφει τον χρόνο που αφιέρωσε ο παίκτης στο παιχνίδι και εμφανίζει τα τελικά αποτελέσματα.

Δημιουργία και Τροποποίηση Πίνακα
Η συνάρτηση `fillSudoku()` χρησιμοποιείται για τη δημιουργία ενός πλήρους, έγκυρου πίνακα Sudoku, με τη βοήθεια αναδρομής και τυχαίας αναδιάταξης αριθμών. Στη συνέχεια, η `removeNumbers()` αφαιρεί αριθμούς από τον πίνακα για να δημιουργηθεί ο γρίφος που πρέπει να λύσει ο παίκτης, ανάλογα με το επίπεδο δυσκολίας.

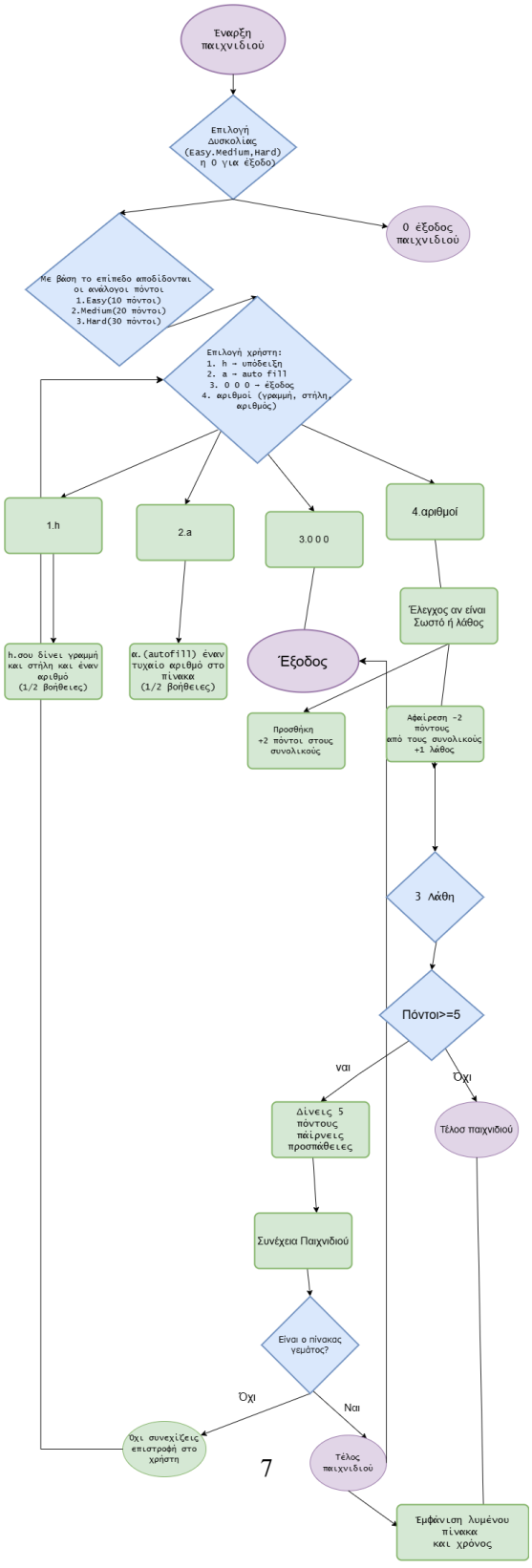
Εμφάνιση και Επαλήθευση
Η `displayBoard()` εμφανίζει τον πίνακα στην κονσόλα με μορφοποίηση που βοηθά τον παίκτη να εντοπίζει τις διακριτές περιοχές του Sudoku. Η συνάρτηση `isSafe()` επαληθεύει αν ένας αριθμός μπορεί να τοποθετηθεί σε μια συγκεκριμένη θέση, σύμφωνα με τους κανόνες του παιχνιδιού (γραμμή, στήλη, τετράγωνο 3x3).

Λογική Παιχνιδιού
Η `playSudoku()` αποτελεί τον βασικό κορμό του παιχνιδιού. Επεξεργάζεται τις εντολές του χρήστη, ελέγχει την εγκυρότητα των απαντήσεων και ενημερώνει τον πίνακα και τους πόντους. Παράλληλα, διαχειρίζεται τα λάθη του παίκτη και του δίνει τη δυνατότητα να αγοράσει επιπλέον προσπάθειες.

Συστήματα Πόντων και Βοήθειας
Η `GamePoints()` ενημερώνει το σκορ του παίκτη βάσει των απαντήσεων, ενώ η `giveHelp()` παρέχει βοήθεια με την εμφάνιση ενός σωστού αριθμού σε κάποιο κενό κελί. Η `Tries()` δίνει τη δυνατότητα αγοράς επιπλέον προσπαθειών (λάθη) με κόστος σε πόντους.

Υποστηρικτικές Συναρτήσεις

Οι `instructions()` και `showDifficultyMenu()` είναι υπεύθυνες για την εμφάνιση οδηγιών και του μενού επιλογών. Η `LevelPoints()` καθορίζει το αρχικό ποσό πόντων που λαμβάνει ο παίκτης ανάλογα με το επιλεγμένο επίπεδο δυσκολίας.



Σχήμα 1: Διάγραμμα ροής της διαδικασίας

4 Code & Implementation

Παρακάτω παρουσιάζεται η δομή του κώδικα του παιχνιδιού SUDOKU.

```
// Function to display difficulty menu
void showDifficultyMenu() {
    cout << "Choose difficulty:\n";
    cout << "1. Easy \n";
    cout << "2. Medium \n";
    cout << "3. Hard \n";
    cout << "0. Exit\n";
}
```

Η παραπάνω συνάρτηση εμφανίζει ένα απλό μενού μέσω της κονσόλας για να επιλέξει ο παίκτης το επίπεδο δυσκολίας. Οι επιλογές 1 έως 3 αντιστοιχούν στα επίπεδα Easy, Medium και Hard, ενώ το 0 χρησιμοποιείται για έξοδο.

Ακολουθεί η συνάρτηση `giveHint`, η οποία εντοπίζει τα κενά κελιά στον πίνακα του Sudoku και παρέχει μια τυχαία βοήθεια:

```
// Function to give a random hint to the user
void giveHint(int board[SIZE][SIZE], int solutionBoard[SIZE][
    SIZE]) {
    vector<pair<int, int>> emptyCells;
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (board[i][j] == 0) {
                emptyCells.push_back({i, j});
            }
        }
    }

    if (!emptyCells.empty()) {
        auto hintCell = emptyCells[rand() % emptyCells.size()];
        int row = hintCell.first;
        int col = hintCell.second;
        cout << "Hint: Try placing a " << solutionBoard[row][
            col]
             << " at row " << row + 1 << ", column " << col + 1
                  << ".\n";
    } else {
        cout << "No hints available. Board is full.\n";
    }
}
```

Η συνάρτηση επιλέγει τυχαία ένα από τα κενά κελιά και υποδεικνύει στον χρήστη τον σωστό αριθμό στη συγκεκριμένη θέση, από τον πίνακα λύσης.

Η συνάρτηση `displayBoard` εμφανίζει τον πίνακα του Sudoku στην κονσόλα, μαζί με τους πόντους ,τις απομένουσες προσπάθειες και βοήθειες του παίκτη, αν αυτές είναι διαθέσιμες.

```
void displayBoard(int board[SIZE][SIZE], int mistakes ,int
    hintsLeft) {
    cout << "\n    1 2 3   4 5 6   7 8 9 \n";
    cout << " -------------------------\n";
    for (int i = 0; i < SIZE; i++) {
        if (i % 3 == 0 && i != 0)
            cout << " -------------------------\n";
        cout << i + 1 << " | ";
        for (int j = 0; j < SIZE; j++) {
            if (j % 3 == 0 && j != 0)
                cout << "| ";
            if (board[i][j] == 0)
                cout << "  ";
            else
                cout << board[i][j] << " ";
        }
        cout << "\n";
    }
    cout << " -------------------------\n";
    cout << "Mistakes: " << mistakes << "/3 | Points: " << C <<
         "\n";
    if (hintsLeft >= 0) {
        cout << "Remaining hints: " << hintsLeft << "/2\n";
    }
}
```

Η συνάρτηση `isSafe` ελέγχει αν ένας αριθμός μπορεί να τοποθετηθεί σε συγκεκριμένο κελί σύμφωνα με τους κανόνες του Sudoku.

```
bool isSafe(int board[SIZE][SIZE], int row, int col, int num) {
    for (int x = 0; x < SIZE; x++) {
        if (board[row][x] == num || board[x][col] == num)
            return false;
    }

    int startRow = row - row % 3;
    int startCol = col - col % 3;

    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (board[startRow + i][startCol + j] == num)
                return false;

    return true;
}
```

Η συνάρτηση επιστρέφει `true` μόνο αν ο αριθμός δεν παραβιάζει κανέναν από τους τρεις βασικούς κανόνες του Sudoku (γραμμή, στήλη, τετράγωνο 3x3).

Η συνάρτηση `fillSudoku` δημιουργεί έναν πλήρη και έγκυρο πίνακα Sudoku με χρήση αναδρομής και backtracking.

```
bool fillSudoku(int board[SIZE][SIZE], int row = 0, int col =
    0) {
    if (row == SIZE - 1 && col == SIZE) return true;
    if (col == SIZE) {
        row++;
        col = 0;
    }
    vector<int> nums = {1,2,3,4,5,6,7,8,9};
    random_shuffle(nums.begin(), nums.end());

    for (int num : nums) {
        if (isSafe(board, row, col, num)) {
            board[row][col] = num;
            if (fillSudoku(board, row, col + 1)) return true;
            board[row][col] = 0;
        }
    }
    return false;
}
```

Η συνάρτηση καλείται αναδρομικά και γεμίζει κάθε κελί με αριθμούς που ελέγχονται για εγκυρότητα.

Η συνάρτηση `fillSudoku` χρησιμοποιείται για να δημιουργήσει έναν πλήρη πίνακα Sudoku με χρήση αναδρομής και backtracking. Πιο συγκεκριμένα:

- Ξεκινά από το κελί (0, 0) και κινείται από αριστερά προς τα δεξιά και από πάνω προς τα κάτω.
- Για κάθε κελί, ανακατεύει (με `random_shuffle`) τους αριθμούς από 1 έως 9 ώστε να υπάρχει ποικιλία και να μην παράγεται πάντα το ίδιο ταμπλό.
- Για κάθε αριθμό, ελέγχει με τη βοήθεια της `isSafe` αν μπορεί να τοποθετηθεί στο κελί.
- Αν ο αριθμός είναι έγκυρος, τον τοποθετεί και καλεί αναδρομικά τη συνάρτηση για το επόμενο κελί.
- Αν φτάσει σε αδιέξοδο, επαναφέρει το κελί στο 0 (backtracking) και δοκιμάζει τον επόμενο αριθμό.
- Όταν τοποθετηθούν επιτυχώς όλοι οι αριθμοί, επιστρέφει `true`, διαφορετικά `false`.

Η χρήση της `random_shuffle` επιτρέπει την παραγωγή διαφορετικών Sudoku κάθε φορά, καθιστώντας τη συνάρτηση κατάλληλη για τη δημιουργία αρχικού ταμπλό με λύση.

Η συνάρτηση `removeNumbers` αφαιρεί αριθμούς από έναν πλήρως λυμένο πίνακα Sudoku, δημιουργώντας το τελικό ταμπλό του παιχνιδιού σύμφωνα με το επιθυμητό επίπεδο δυσκολίας.

```
void removeNumbers(int board[SIZE][SIZE], int count) {
    vector<pair<int, int>> positions;
    for (int i = 0; i < SIZE; i++)
        for (int j = 0; j < SIZE; j++)
            positions.push_back({i, j});

    random_shuffle(positions.begin(), positions.end());

    int removeCount = SIZE * SIZE - count;
    for (int i = 0; i < removeCount; i++) {
        int r = positions[i].first;
        int c = positions[i].second;
        board[r][c] = 0;
    }
}
```

Η συνάρτηση `removeNumbers` χρησιμοποιείται για να ”αδειάσει” επιλεγμένα κελιά από έναν πλήρως λυμένο πίνακα Sudoku, με στόχο τη δημιουργία του ταμπλό που θα παρουσιαστεί στον παίκτη.

Πιο αναλυτικά:

- Δημιουργεί έναν πίνακα με τις θέσεις όλων των κελιών του ταμπλό.
- Ανακατεύει τις θέσεις τυχαία, ώστε να επιλέγονται διαφορετικά κελιά κάθε φορά.
- Υπολογίζει πόσα κελιά πρέπει να αδειάσουν, με βάση το πλήθος `count`, που αντιστοιχεί στον αριθμό των κελιών που *πρέπει να μείνουν* στο ταμπλό.
- Τέλος, θέτει σε `0` (κενό) τα επιλεγμένα κελιά.

Η παράμετρος `count` ρυθμίζεται ανάλογα με τη δυσκολία: π.χ. περισσότερα γνωστά κελιά για *Easy*, λιγότερα για *Hard*.

Η συνάρτηση `isBoardFull` ελέγχει αν ο πίνακας Sudoku έχει συμπληρωθεί πλήρως, δηλαδή αν όλα τα κελιά περιέχουν τιμές διαφορετικές του μηδενός.

```
bool isBoardFull(int board[SIZE][SIZE]) {
    for (int i = 0; i < SIZE; ++i)
        for (int j = 0; j < SIZE; ++j)
            if (board[i][j] == 0)
                return false;
    return true;
}
```

vspace1em

Η συνάρτηση `isBoardFull` επιστρέφει `true` αν όλα τα κελιά του πίνακα έχουν συμπληρωθεί (δηλαδή δεν υπάρχει καμία θέση με τιμή 0), διαφορετικά επιστρέφει `false`.

Αναλυτικά:

- Κάνει επανάληψη σε κάθε κελί του πίνακα.
- Αν βρεθεί κάποιο κελί με τιμή `0`, επιστρέφει άμεσα `false`, καθώς αυτό σημαίνει πως ο πίνακας δεν είναι πλήρης.
- Αν ολοκληρωθεί ο έλεγχος χωρίς να εντοπιστεί κενό κελί, τότε επιστρέφεται `true`.

Η συνάρτηση αυτή είναι χρήσιμη για να ελέγχεται αν ο παίκτης έχει ολοκληρώσει το Sudoku ή για να καθοριστεί πότε πρέπει να τερματιστεί το παιχνίδι.

Συνάρτηση `autoFillOneCell`

Η συνάρτηση `autoFillOneCell` προσφέρει βοήθεια στον παίκτη του Sudoku όταν έχει κολλήσει. Ελέγχει τον πίνακα παιχνιδιού για το πρώτο κενό κελί (δηλαδή κελί με τιμή `0`) και το συμπληρώνει αυτόματα με τη σωστή τιμή από τον πίνακα λύσης `solutionBoard`.

Με αυτόν τον τρόπο, ο παίκτης λαμβάνει μια υπόδειξη για την επόμενη σωστή κίνηση. Αν ο πίνακας είναι ήδη πλήρης, εμφανίζεται σχετικό μήνυμα. Η συνάρτηση συμπληρώνει μόνο **ένα** κελί κάθε φορά.

```
void autoFillOneCell(int board[SIZE][SIZE], int solutionBoard[
    SIZE][SIZE]) { //dinei pragmatikh voitheia
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (board[i][j] == 0) {
                board[i][j] = solutionBoard[i][j];
                cout << "Auto-filled cell at row " << i + 1 <<
                    ", column " << j + 1
                     << " with value " << solutionBoard[i][j]
                        << ".\n";
                return;
            }
        }
    }
    cout << "Board is already full.\n";
}
```

Η συνάρτηση `playSudoku` διαχειρίζεται ολόκληρη τη ροή του παιχνιδιού Sudoku από την πλευρά του παίκτη. Παρέχει τη δυνατότητα εισαγωγής αριθμών, χρήσης βοήθειας (hint), ή αυτόματης συμπλήρωσης ενός κελιού (auto-fill).

```
void playSudoku(int board[SIZE][SIZE], int solutionBoard[SIZE][
    SIZE]) {
    int row, col, num;
    int mistakes=0,hintsLeft=2;

    displayBoard(board, mistakes,hintsLeft);

    while (!isBoardFull(board)) {

  cout << "\nEnter row (1-9), column (1-9), and number (1-9),
      or type 'h' for hint, 'a' for auto-fill, or '0 0 0' to
      exit: ";
  string input;
  cin >> ws;
  getline(cin, input);

  if (input == "h") {
      if (hintsLeft > 0) {
          giveHint(board, solutionBoard);
          hintsLeft--;
      } else {
          cout << "No hints left.\n";
      }
      displayBoard(board, mistakes, hintsLeft);
      continue;
    }
  else if (input == "a") {
      if (hintsLeft > 0) {
          autoFillOneCell(board, solutionBoard);
          hintsLeft--;
      } else {
          cout << "No auto-fills left.\n";
      }
      displayBoard(board, mistakes, hintsLeft);
      continue;
  }
  else {
      istringstream iss(input);
      if (!(iss >> row >> col >> num)) {
          cout << "Invalid input. Please enter either 'h', 'a',
               or 3 numbers.\n";
          continue;
      }
  }

        if (row == 0 && col == 0 && num == 0) {
        cout << "You exited the game.\n";
        break;
    }
```

```
46
47
48          row--;
49          col--;
50
51          if (row >= 0 && row < SIZE && col >= 0 && col < SIZE &&
                num >= 1 && num <= 9) {
52              if (board[row][col] == 0) {
53          GamePoints(board, solutionBoard, row, col, num);
54          if (solutionBoard[row][col] == num) {
55              board[row][col] = num;
56              displayBoard(board,mistakes,hintsLeft);
57          }
58          else {
59              mistakes++;
60              cout << "Mistakes:" << mistakes << "/3\n";
61              displayBoard(board,mistakes,hintsLeft);
62              if (mistakes == 3) {
63                bool bought = Tries(mistakes);
64                cout << "Remaining chances: " << mistakes << "/3\
                      n";
65              displayBoard(board,mistakes,hintsLeft);
66              if (!bought) { break;}
67            }
68          }
69        }
70        else {
71                cout << "Cell already filled.\n";
72        }
73        }
74      else{
75                cout << "Invalid input. Try again.\n";
76      }
77    }
78
79      if (isBoardFull(board))
80            cout << "Congratulations! You've completed the Sudoku
                  !\n";
81        cout << "\nSolution Board:\n";
82        displayBoard(solutionBoard,mistakes,hintsLeft);
83  }
```

Η συνάρτηση αυτή είναι χρήσιμη για να ελέγχεται αν ο παίκτης έχει ολοκληρώσει το Sudoku ή για να καθοριστεί πότε πρέπει να τερματιστεί το παιχνίδι.

Η συνάρτηση `playSudoku` διαχειρίζεται ολόκληρη τη ροή του παιχνιδιού Sudoku από την πλευρά του παίκτη. Παρέχει τη δυνατότητα εισαγωγής αριθμών, χρήσης βοήθειας (hint) ή αυτόματης συμπλήρωσης ενός κελιού (auto-fill). Παρακολουθεί τις ευκαιρίες (chances) και τις διαθέσιμες βοήθειες (hints), και ενημερώνει τον παίκτη για

την πρόοδο του παιχνιδιού.

Αναλυτικά, η συνάρτηση υλοποιεί τα εξής:

- Εμφανίζει την τρέχουσα κατάσταση του πίνακα Sudoku, μαζί με τις υπολειπόμενες ευκαιρίες και βοήθειες.
- Εκτελεί έναν βρόχο, ο οποίος συνεχίζεται όσο υπάρχουν κενά κελιά στον πίνακα.
- Σε κάθε επανάληψη, ζητά από τον χρήστη είσοδο:
  - Αν η είσοδος είναι `h`, γίνεται προσπάθεια παροχής βοήθειας (αν υπάρχουν διαθέσιμες βοήθειες).
  - Αν η είσοδος είναι `a`, γίνεται αυτόματη συμπλήρωση ενός κελιού (αν υπάρχουν διαθέσιμες βοήθειες).
  - Αν δοθεί είσοδος της μορφής `row col num`, γίνεται προσπάθεια τοποθέτησης του αριθμού στο αντίστοιχο κελί:
    - Αν η τοποθέτηση είναι σωστή, ο αριθμός καταχωρείται.
    - Αν είναι λανθασμένος, μειώνεται ο αριθμός των ευκαιριών.
- Αν εξαντληθούν οι ευκαιρίες, το παιχνίδι τερματίζεται με ήττα.
- Αν συμπληρωθούν σωστά όλα τα κελιά, το παιχνίδι ολοκληρώνεται με επιτυχία.
- Στο τέλος, εμφανίζεται ο πίνακας λύσης για αναφορά.

Η συνάρτηση αυτή αποτελεί τον βασικό «βρόχο παιχνιδιού» και συντονίζει την αλληλεπίδραση του χρήστη με τον πίνακα Sudoku. Είναι υπεύθυνη για την πρόοδο του παιχνιδιού, την ορθή διαχείριση των επιλογών του παίκτη και τη συνολική εμπειρία χρήσης.

Η συνάρτηση `main` αποτελεί το σημείο εκκίνησης του προγράμματος. Αρχικοποιεί τον πίνακα Sudoku, επιτρέπει στον χρήστη να επιλέξει επίπεδο δυσκολίας, ξεκινά το παιχνίδι και στο τέλος υπολογίζει και εμφανίζει τον συνολικό χρόνο που χρειάστηκε ο παίκτης.

```
1  int main() {
2      int ep;
3      int board[SIZE][SIZE] = {0};
4      int solutionBoard[SIZE][SIZE] = {0};
5      int visibleNumbers;
6      srand(time(0));
7
8      using namespace std::chrono;
9      auto start = steady_clock::now();
10
11     cout << "==============================\n";
12     cout << " \tWelcome to Sudoku!\n";
```

```
    cout << "============================\n";
    char ans;
    cout << "Do you want to see the instructions?(y/n): ";//
        rctaei gia emfanisi odigion
    cin >> ans;
    ans=tolower(ans);
    while(ans!='y' && ans!='n'){
      cout<<"Wrong input.Enter again(y/n):";
      cin >> ans;
      ans=tolower(ans);
  }
    if (ans == 'y') {
        instructions();
    }
    showDifficultyMenu();
    cin >> ep;

    while (ep < 0 || ep > 3) {
        cout << "Invalid input. Please enter a number between 0
            and 3 (0 for exit): ";
        showDifficultyMenu();
        cin >> ep;
    }

    LevelPoints(ep);
    if (ep == 1)
        visibleNumbers = 39;
    else if (ep == 2)
        visibleNumbers = 35;
    else if (ep == 3)
        visibleNumbers = 29;
    else {
        auto end = steady_clock::now();
    auto duration = duration_cast<seconds>(end - start).count()
        ;
    int minutes = duration / 60;
    int seconds = duration % 60;
    cout << "You exited the game.\n";
    displayTime(duration);
    return 0;
    }

    fillSudoku(board);
    copy(&board[0][0], &board[0][0] + SIZE * SIZE, &
        solutionBoard[0][0]); // Save full solution
    removeNumbers(board, visibleNumbers);

    switch (ep) {
        case 1:
      cout << "\n\tEasy level.\n"; break;
```

```
        case 2:
      cout << "\n\tMedium level.\n"; break;
        case 3:
      cout << "\n\tHard level.\n"; break;
    }


    playSudoku(board, solutionBoard);

    auto end = steady_clock::now();
  auto duration = duration_cast<seconds>(end - start).count();

  displayTime(duration);

  cout << "Your total points are: " << C << "\n";
    return 0;
}
```

Αναλυτικά:

- Εμφανίζει το κεντρικό μήνυμα και το μενού επιλογής δυσκολίας.
- Αν ο παίκτης επιλέξει 0, το πρόγραμμα τερματίζει εμφανίζοντας τον χρόνο.
- Με βάση τη δυσκολία, υπολογίζει πόσοι αριθμοί θα παραμείνουν εμφανείς στον πίνακα.
- Δημιουργεί έναν ολοκληρωμένο πίνακα Sudoku και στη συνέχεια αφαιρεί αριθμούς για να τον κάνει κατάλληλο για επίλυση.
- Καλεί τη `playSudoku` για να ξεκινήσει το παιχνίδι.
- Μετρά και εμφανίζει τον συνολικό χρόνο που χρειάστηκε ο παίκτης.

Η συνάρτηση `instructions` εμφανίζει τις οδηγίες του Sudoku στη κονσόλα,καθώς και το σύστημα πόντων που βασίζεται στο επίπεδο δυσκολίας που έχει επιλέξει ο χρήστης.

```
void instructions(){
 cout<<"According to the difficulty level you will choose,you
     will start with some points:\n";
 cout<<"Levels:\n";
 cout << "1. Easy   - 10 points\n";
   cout << "2. Medium - 20 points\n";
   cout << "3. Hard   - 30 points\n\n";
   cout << "- Fill the grid so that each number from 1 to 9\n"
       ;
   cout << "    appears exactly once in every row, column, and
        3x3 box.\n";
   cout << "- To enter a number, type: row column value (e.g.
       3 4 7).\n";
```

```
    cout << "- You can use 'h' to reveal a correct value (for free).\n";
    cout << "- You can use 'a' to fill in all obvious values (for free).\n";
    cout << "- If you run out of attempts, you can buy 3 extra attempts for 5 points.\n";
    cout << "- You can exit the game whenever you want.\n";

    cout << "Scoring:\n";
  cout << "- You gain 2 points for every correct number you enter.\n";
  cout << "- You lose 2 points for every incorrect number.\n";
}
```

Η συνάρτηση `giveHelp` χρησιμοποιείται για την παροχή βοήθειας στον χρήστη,όταν αυτός θέλει να βοηθηθεί για τη συμπλήρωση του πίνακα.

```
bool giveHelp(int board[SIZE][SIZE], int solutionBoard[SIZE][SIZE]) {
    for (int i = 0; i < SIZE; ++i) {
        for (int j = 0; j < SIZE; ++j) {
            if (board[i][j] == 0) {
                board[i][j] = solutionBoard[i][j];
                cout << "Help used: Cell (" << i+1 << "," << j+1 << ") filled with " << solutionBoard[i][j] << ".\n";
                return true;
            }
        }
    }
    return false;
}
```

Η συνάρτηση `GamePoints` διαχειρίζεται τη διαδικασία που υπάρχει με τους πόντους,δηλαδή:

- με κάθε σωστή τοποθέτηση του χρήστη αυξάνει τους πόντους κατά 2.
- με κάθε λάθος τοποθέτηση του χρήστη μειώνει τους πόντους κατά 2.

```
void GamePoints(int board[SIZE][SIZE], int solutionBoard[SIZE][SIZE], int row, int col, int num){
    if (row >= 0 && row < SIZE && col >= 0 && col < SIZE && num >= 1 && num <= 9) {
        if (board[row][col] == 0) {
            if (solutionBoard[row][col] == num) {
                C += 2;
            } else {
                C -= 2;
```

```
            }
        }
    }
}
```

Η συνάρτηση `Tries` ελέγχει αν ο χρήστης έχει αρκετούς πόντους για να αγοράσει 3 επιπλέον προσπάθειες άμα έχει ξεμείνει.Αν έχει τουλάχιστον 5 πόντους του γίνεται ερώτηση και αν επιλέξει ναι παίρνει τις επιπλέον προσπάθειες.

```
bool Tries(int& chances) {
    if (C < 5) {
        cout << "Sorry, you don't have enough points to buy
            more chances.\n";
        return false;
    }

    char an;
    do {
        cout << "Would you like to buy 3 more chances for 5
            point? (y/n): ";
        cin >> an;
        an = tolower(an);
        if (an != 'y' && an != 'n') {
            cout << "Invalid input. Please type 'y' or 'n'.\n";
        }
    } while (an != 'y' && an != 'n');

    if (an == 'y') {
        chances =0;
        C -= 5;
        cout << "You just bought 3 more chances. Remaining
            points: " << C << "\n";
        return true;
    } else {
        cout << "You chose not to buy more chances.\n";
        return false;
    }
}
```

Η λειτουργία της συνάρτησης `LevelPoints` είναι οτι κατά την επιλογή επιπέδου απο τον χρήστη,αναθέτει τους ανάλογους πόντους που του παρέχονται.

```
void LevelPoints(int l){
  if (l==1){
    C=C+10;
  }
  else if (l==2){
    C=C+20;
  }
```

```
  else if (l==3){
    C=C+30;
  }
}
```

Τέλος η συνάρτηση `displayTime` υπολογίζει τα λεπτά και θα δευτερόλεπτα που πέρασαν απο την έναρξη του παιχνιδιού μέχρι την ολοκλήρωσή του ή την επιθυμητή αποχώρηση του χρήστη.Εμφανίζεται σε μορφή mm:ss όπου και τα λεπτά και τα δευτερόλεπτα εμφανίζονται σε μορφή δύο ψηφίων.

```
void displayTime(int duration) {
    int minutes = duration / 60;
    int seconds = duration % 60;

    cout << "Time taken: "
         << setw(2) << setfill('0') << minutes << ":"
         << setw(2) << setfill('0') << seconds << "\n";
}
```

5 Κώδικας Υλοποίηση

5.1 Πως υλοποιήθηκε η βασική λειτουργικότητα

Η βασική λειτουργικότητα του παιχνιδιού Sudoku υλοποιήθηκε με χρήση δομημένου προγραμματισμού στην C++. Το πρόγραμμα βασίζεται σε μια σειρά από επιμέρους συναρτήσεις, καθεμία από τις οποίες επιτελεί μια σαφώς καθορισμένη λειτουργία, όπως η δημιουργία του πίνακα, η εμφάνιση του παιχνιδιού στην οθόνη, η επεξεργασία των κινήσεων του παίκτη και ο έλεγχος της εγκυρότητας των αριθμών. Η συνάρτηση main() λειτουργεί ως το κεντρικό σημείο ελέγχου, καθώς διαχειρίζεται τη ροή του παιχνιδιού, ξεκινώντας από την επιλογή του επιπέδου δυσκολίας και καταλήγοντας στην έναρξη και ολοκλήρωση του παιχνιδιού με καταγραφή του χρόνου και της επίδοσης του παίκτη. Κάθε μέρος του παιχνιδιού έχει υλοποιηθεί ως ανεξάρτητη συνάρτηση, κάτι που διευκολύνει την αναγνωσιμότητα και τη συντήρηση του κώδικα.

5.2 Ποιοί είναι οι σημαντικοί αλγόριθμοι ή δομές δεδομένων που χρησιμοποιήθηκαν

Στο παιχνίδι έχουν χρησιμοποιηθεί βασικές δομές δεδομένων όπως δισδιάστατοι πίνακες (`int board[9][9]`), οι οποίοι χρησιμοποιούνται τόσο για την αποθήκευση του αρχικού πίνακα Sudoku όσο και για την αποθήκευση της λύσης του. Ένας από τους πιο σημαντικούς αλγόριθμους που υλοποιήθηκαν είναι η αναδρομική επίλυση του πίνακα μέσω της συνάρτησης `fillSudoku()`, η οποία χρησιμοποιεί τεχνικές *backtracking* για να δημιουργήσει μια πλήρη και σωστή λύση του Sudoku.

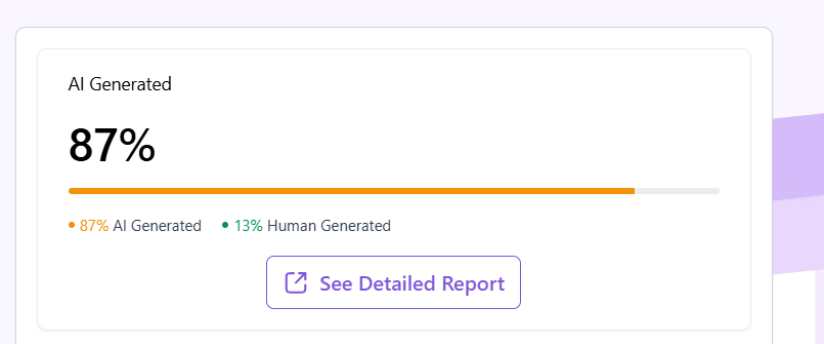
Επιπλέον, η συνάρτηση `removeNumbers()` αφαιρεί αριθμούς από τη λύση με τυχαίο τρόπο, δημιουργώντας έναν γρίφο ανάλογα με το επιλεγμένο επίπεδο δυσκολίας. Η `isSafe()` ελέγχει εάν ένας αριθμός μπορεί να τοποθετηθεί σε μια θέση χωρίς να παραβιάζει τους κανόνες του παιχνιδιού, διασφαλίζοντας έτσι την εγκυρότητα κάθε κίνησης. Αυτοί οι αλγόριθμοι, αν και σχετικά απλοί, είναι ουσιώδεις για την ορθή λειτουργία και τη συνολική πρόκληση του παιχνιδιού.

6 Αποτελέσματα και Demo

Εφαρμογή στην οποία είναι ανεβασμένα τα βιντεάκια είναι Dropbox.
Εδώ μπορείτε να πατήσετε και να δείτε τα βιντεάκια του Demo
Το demo περιλαμβάνει τα εξής βίντεο (αριθμημένα από 1 έως 7):

1) Βίντεο με το Μενού
2) Επίπεδα δυσκολίας: *Easy*, *Medium*, *Hard*
3) Χρήση της λειτουργίας *Hint*
4) Χρήση της λειτουργίας *Auto-fill*
5) Αντιμετώπιση *mistakes* (με αρκετούς πόντους για αγορά ευκαιριών)
6) Αντιμετώπιση *mistakes* (χωρίς αρκετούς πόντους)
7) Έξοδος από το παιχνίδι

7 Σύγκριση με Κώδικα που Δημιουργήθηκε από Τεχνητή Νοημοσύνη



Σχήμα 2: Σύκριση με AI

8 Συμπεράσματα και Μαθήματα που Αποκομίστηκαν

Κατά τη διάρκεια της υλοποίησης της παρούσας ομαδικής εργασίας, αποκτήσαμε πολύτιμη εμπειρία τόσο σε τεχνικό όσο και σε συνεργατικό επίπεδο. Η δημιουργία του παιχνιδιού Sudoku μάς έδωσε την ευκαιρία να εφαρμόσουμε στην πράξη γνώσεις που έχουμε διδαχθεί στη θεωρία, να ενισχύσουμε τις δεξιότητές μας στον προγραμματισμό και να εμβαθύνουμε στη λογική σχεδίασης λογισμικού.

Αρχικά, κατανοήσαμε σε βάθος τη χρήση δισδιάστατων πινάκων, τις δομές ελέγχου, αλλά και την έννοια της αναδρομής, μέσα από την κατασκευή του αλγορίθμου συμπλήρωσης του πίνακα Sudoku. Η επίλυση του προβλήματος απαιτούσε τη χρήση στρατηγικής και τη διαχείριση πολύπλοκων συνθηκών εγκυρότητας, ενισχύοντας έτσι τη λογική μας σκέψη και την ικανότητά μας στην επίλυση προβλημάτων.

Παράλληλα, δώσαμε έμφαση στη φιλικότητα της εφαρμογής προς τον χρήστη, προσθέτοντας λειτουργίες όπως οι οδηγίες, τα hints, τα auto-fills, το σύστημα πόντων και τα επίπεδα δυσκολίας. Μέσα από αυτά, κατανοήσαμε καλύτερα τη σημασία της σωστής διεπαφής χρήστη ακόμα και σε περιβάλλον κονσόλας, καθώς και τη σημασία της σαφούς επικοινωνίας με τον χρήστη μέσω κατάλληλων μηνυμάτων.

Η εργασία μάς έδωσε επίσης την ευκαιρία να συνεργαστούμε αποτελεσματικά ως ομάδα, μοιράζοντας αρμοδιότητες και ελέγχοντας τακτικά την πορεία του έργου. Μάθαμε να συντονιζόμαστε, να αξιοποιούμε διαφορετικές ιδέες και να αντιμετωπίζουμε από κοινού τις προκλήσεις που προέκυπταν κατά τη διάρκεια της ανάπτυξης.

Επιπλέον, μέσα από τη σύγκριση του δικού μας κώδικα με έτοιμες λύσεις που μπορεί να παραχθούν μέσω εργαλείων Τεχνητής Νοημοσύνης, αντιληφθήκαμε ότι, ενώ η AI μπορεί να προσφέρει ταχύτητα και αποδοτικότητα, η διαδικασία της χειροκίνητης υλοποίησης είναι μοναδικά εκπαιδευτική. Μας προσέφερε ουσιαστική κατανόηση των μηχανισμών που βρίσκονται πίσω από τη λειτουργία ενός παιχνιδιού Sudoku και μας επέτρεψε να δούμε πώς δομείται και οργανώνεται μια πραγματική εφαρμογή από το μηδέν.

Τέλος, αισθανθήκαμε μεγάλη ικανοποίηση βλέποντας το τελικό αποτέλεσμα μιας λειτουργικής εφαρμογής που αναπτύξαμε εξ ολοκλήρου μόνοι μας. Το έργο αυτό ενίσχυσε την αυτοπεποίθησή μας ως μελλοντικοί προγραμματιστές και μας ενέπνευσε να συνεχίσουμε να δημιουργούμε και να εξελισσόμαστε στον τομέα της ανάπτυξης λογισμικού.