ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

Πρόγραμμα Αναζήτησης UCS και A* - Κατασκευή Παιγνίου.

Team

Γκόβαρης Χρήστος-Γρηγόριος
Κωτόπουλος Βασίλειος
Σπανού Μαρία

Πρόγραμμα Αναζήτησης με UCS και A*

**Η παρούσα άσκηση υλοποιήθηκε σε Java, σε 3 κλάσεις.*

Η πρώτη εργαστηριακή μας άσκηση, αποτελεί μία παραλλαγή του γνωστού προβλήματος 8-puzzle, όπου εκτός από τις συνήθεις μετακινήσεις (οριζόντιες και κατακόρυφες) σε μία γειτονική κενή θέση, επιτρέπεται και η διαγώνια μετακίνηση.

Στόχος μας είναι να βρούμε την ελάχιστη ακολουθία ενεργειών από μία αρχική κατάσταση (ΑΚ) προς μία τελική κατάσταση (ΤΚ).

Για τον σκοπό αυτόν, θα υλοποιήσουμε δύο μεθόδους αναζήτησης, την μέθοδο Αναζήτησης Ομοιόμορφου Κόστους (UCS) και την Αναζήτηση A* χρησιμοποιώντας μία καλύτερη δυνατή ευρετική συνάρτηση h(n).

Χρονοδιάγραμμα 1ης Εργαστηριακής Άσκησης

Έκδοση	Ημερομηνία	Progress
1.0	27/3/2024	Ανακοίνωση 1ης εργαστηριακής άσκησης
2.0	29/3 - 15/4	Υλοποίηση άσκησης
3.0	16/5/2024	Σχόλια στον κώδικα
4.0	17/5/2024	Υλοποίηση Report 1ης εργαστηριακής άσκησης

Ανάλυση Χαρακτηριστικών

Η εργασία υλοποιήθηκε σε ένα μηχάνημα (Lenovo Ideapad 3), με τα εξής χαρακτηριστικά πυρήνα:

- AMD Ryzen 7 (7730U)
- 8 CPU cores
- 16 Threads
- Boost Clock up to 4.5GHz
- Base Clock 2.0GHz
- CPU Socket FP6
- Intergraded Graphics (Radeon Graphics)

Επιπλέον, η εργασία υλοποιήθηκε σε **Windows 11 Home**, χρησιμοποιώντας το **Visual Studio**.

PuzzleSolver.java:

Στην παρούσα κλάση, έχουμε ορίσει το πεδίο **private int[][] goalState**, το οποίο θέτει την τελική κατάσταση, όπως αναγράφεται στην εκφώνηση.

Έπειτα, έχουμε υλοποιήσει τις εξής μεθόδους:

public void solve(): Η μέθοδος αυτή, παίρνει σαν όρισμα την αρχική κατάσταση, χρησιμοποιεί είτε τον αλγόριθμο UCS είτε τον αλγόριθμο A* ανάλογα με την τιμή της παραμέτρου aStar. Η solve ελέγχει αν η αρχική κατάσταση είναι έγκυρη (valid) και δημιουργεί μία PriorityQueue, η οποία είναι φτιαγμένη με βάση το κόστος των καταστάσεων. Η επιλογή του κόστους γίνεται ανάλογα με την παράμετρο aStar, η οποία καθορίζει αν θα χρησιμοποιηθεί ο ευρετικός αλγόριθμος A*. Δημιουργεί ένα σύνολο (explored) για να αποθηκεύσει τις καταστάσεις που έχουν ήδη εξερευνηθεί και ένα σύνολο (inFrontier), το οποίο απεικονίζει τα στοιχεία που βρίσκονται στην PriorityQueue για γρήγορο έλεγχο των υπάρχοντων καταστάσεων. Υπάρχει ένας βρόχος (while loop), που εκτελείται έως ότου να αδειάσει η ουρά. Σε κάθε επανάληψη, αφαιρεί την πρώτη κατάσταση από την ουρά και ελέγχει αν αυτή είναι η τελική κατάσταση. Αν ναι, τότε εκτυπώνει την λύση και τερματίζει, διαφορετικά προσθέτει την κατάσταση στο σύνολο explored και παράγει όλες τις πιθανές καταστάσεις που προκύπτουν από την τρέχουσα μετακίνηση, προσθέτοντάς τις στην ουρά μόνο αν δεν έχουν εξερευνηθεί ή δεν βρίσκονται στην ουρά. Στην συνέχεια, αν η ουρά είναι κενή και δεν έχει βρεθεί λύση, εκτυπώνει ένα μήνυμα σφάλματος.

private boolean isValidState(): Η μέθοδος αυτή, παίρνει ως όρισμα έναν πίνακα που συμβολίζει την κατάσταση. Χρησιμοποιείται για να επιβεβαιώσει την εγκυρότητα μίας κατάστασης στο 8-puzzle, δηλαδή αν περιέχει όλους τους αριθμούς από το 0 έως και το 8 ακριβώς μία φορά. Δημιουργούμε ένα σύνολο tiles, τύπου HashSet, για την αποθήκευση των αριθμών των πλακιδίων που περιέχονται στην κατάσταση του παζλ. Στην συνέχεια, ελέγχει κάθε στοιχείο του πίνακα state και το προσθέτει μέσα στο σύνολο tiles. Έπειτα, ελέγχει αν το tiles περιέχει όλους τους αριθμούς από το 0 έως το 8, που αντιστοιχούν στα πλακίδια του παζλ. Αν κάποιος από αυτούς τους αριθμούς λείπει, η μέθοδος επιστρέφει false (δείχνοντας ότι η κατάσταση δεν είναι έγκυρη), διαφορετικά η μέθοδος επιστρέφει true (δείχνοντας ότι η κατάσταση μπορεί να χρησιμοποιηθεί για την επίλυση του παζλ).

private void printSolution(): Η μέθοδος αυτή, εκτυπώνει το μονοπάτι της λύσης, από την αρχική κατάσταση έως και την τελική, καθώς και το κόστος και το βάθος της τελικής κατάστασης. Αρχικά, λαμβάνει το μονοπάτι (από την ΑΚ έως και την ΤΚ), χρησιμοποιώντας την μέθοδο getPathFromStart() της κατάστασης state που έχουμε δώσει σαν όρισμα. Στην συνέχεια, εκτυπώνει το μονοπάτι που ακολουθήθηκε,

παραθέτοντας τις καταστάσεις σε κάθε βήμα του μονοπατιού. Για κάθε κατάσταση, χρησιμοποιεί την μέθοδο getState() για να λάβει τον πίνακα κατάστασης και εκτυπώνει κάθε γραμμή του πίνακα. Στο τέλος κάθε κατάστασης, τυπώνει ένα κενό για να ξεχωρίσει τις διαφορετικές καταστάσεις που ακολουθούν. Επίσης, εμφανίζει το κόστος της τελικής κατάστασης όπως και το βάθος της, χρησιμοποιώντας τις μεθόδους getCost() και getDepth() αντίστοιχα, της κατάστασης state.

private int findZeroRow(): Η μέθοδος αυτή, ψάχνει να βρει το 0 στις γραμμές της κατάστασης.

private int findZeroCol(): Η μέθοδος αυτή, ψάχνει να βρει το 0 στις στήλες της κατάστασης.

PuzzleState.java:

Στην παρούσα κλάση, έχουμε ορίσει τα πεδία:

private int[][] state: Το πεδίο αυτό, απεικονίζει την τρέχουσα κατάσταση του παζλ.

private PuzzleState parent: Το πεδίο αυτό, απεικονίζει την θέση του κενού στοιχείου της κατάστασης

private int zeroRow, zeroCol: Το πεδίο αυτό, απεικονίζει την θέση του κενού στοιχείου της κατάστασης.

private int cost: Το πεδίο αυτό, απεικονίζει το κόστος.

private int depth: απεικονίζει τον αριθμό των κινήσεων (moves) για να φτάσει σε μία συγκεκριμένη κατάσταση.

Έπειτα, έχουμε υλοποιήσει τις εξής μεθόδους:

public List<PuzzleState> generateSuccessors(): Η μέθοδος αυτή, δημιουργεί και επιστρέφει τις διαδοχικές καταστάσεις, μετακινώντας το κενό κελί προς διαφορετικές κατευθύνσεις. Αναλυτικότερα, δημιουργεί μία κενή λίστα που θα περιέχει τις διαδοχικές καταστάσεις. Έπειτα, ορίζει έναν πίνακα (directions) που περιέχει τις 8 δυνατές κατευθύνσεις κίνησης για το κενό κελί (πάνω, κάτω, δεξιά, αριστερά και τις 4 διαγώνιες μετακινήσεις). Στην συνέχεια, επαναλαμβάνει μέσω του πίνακα directions, όπου για κάθε κατεύθυνση υπολογίζει την νέα σειρά και στήλη του κενού κελιού, λαμβάνοντας υπόψη την τρέχουσα θέση του κενού κελιού (zeroRow και zeroCol) και την κατεύθυνση. Ελέγχει αν η νέα σειρά και στήλη, βρίσκεται εντός των ορίων του πίνακα 3x3. Αν ναι, τότε δημιουργεί έναν νέο πίνακα κατάστασης (newState), ο οποίος είναι ένα deep copy του τρέχοντος πίνακα κατάστασης. Ανταλλάσσει τις τιμές του κενού κελιού, με αυτές της νέας θέσης, προσομοιώνοντας. Τέλος, δημιουργεί ένα αντικείμενο PuzzleState, με τον νέο πίνακα κατάσταση, τις νέες συντεταγμένες του νέου κελιού, την τρέχουσα κατάσταση ως γονέα, το κόστος του παιχνιδιού (αυξημένο κατά 1) και το βάθος (αυξημένο κατά 1). Προσθέτει αυτό το νέο αντικείμενο στην λίστα των διαδοχικών καταστάσεων, που πρόκειται να επιστραφούν. Αυτή η μέθοδος, είναι κρίσιμη για την εύρεση της λύσης του παζλ, καθώς δημιουργεί τις διαδοχικές καταστάσεις που ελέγχονται και αξιολογούνται από τον αλγόριθμο αναζήτησης.

private int[][] deepCory: Αυτή η μέθοδος, δέχεται έναν πίνακα δύο διαστάσεων (int[][] original) και δημιουργεί ένα βαθύ αντίγραφο του πίνακα αυτού. Αρχικά, δημιουργείται ένας νέος πίνακας copy, με τον ίδιο αριθμό σειρών, με τον πίνακα original. Στην συνέχεια, για κάθε σειρά του original, δημιουργείται ένας νέος πίνακας, με τον ίδιο αριθμό στοιχείων, όπως και η αρχική σειρά χρησιμοποιώντας την μέθοδο

Arrays.copyOf(). Τέλος, στον νέο πίνακα αντιγράφονται όλες οι τιμές του original, σειρά προς σειρά, στον πίνακα copy. Το αποτέλεσμα, είναι ένας νέος πίνακας, που περιέχει ακριβώς τα ίδια στοιχεία με τον αρχικό (αλλά είναι ανεξάρτητος από αυτόν), διασφαλίζοντας ένα βαθύ αντίγραφο των δεδομένων.

public boolean isGoal(): Η μέθοδος αυτή, χρησιμοποιείται για να ελέγξει αν η τρέχον κατάσταση του προβλήματος που αναπαρίσταται από έναν πίνακα διαστάσεων int[][] this.state, είναι ο στόχος που ορίζεται από τον πίνακα goalState. Η μέθοδος, χρησιμοποιεί την στατική μέθοδο Arrays.deepEquals, η οποία συγκρίνει δύο πολυδιάστατους πίνακες με βάθος, ελέγχοντας αν τα περιεχόμενά τους είναι ίδια. Έτσι η μέθοδος isGoal(), επιστρέφει true (αν η τρέχουσα κατάσταση του προβλήματος είναι ίδια με τον στόχο), ενώ διαφορετικά επιστρέφει false. Αυτό επιτρέπει στο πρόγραμμα να ελέγχει αν έχει επιτευχθεί ο επιθυμητός στόχος ή όχι (βάση των δύο πινάκων κατάστασης).

public List<PuzzleState> getPathFromStart: Η μέθοδος αυτή, δημιουργεί μία λίστα (path) που περιέχει το μονοπάτι από την αρχική κατάσταση του παζλ προς την τρέχουσα κατάσταση. Αρχικά, δημιουργείται μία κενή λίστα path και η μεταβλητή current, αρχικοποιείται με την τρέχουσα κατάσταση του παζλ. Σε έναν βρόχο while, η μέθοδος εξετάζει αν η τρέχουσα κατάσταση δεν είναι null. Αν δεν είναι, προσθέτει την τρέχουσα κατάσταση στην αρχή της λίστας path, χρησιμοποιώντας την μέθοδο add(0, current), που εισάγει το στοιχείο στην αρχή της λίστας. Στην συνέχεια, η μεταβλητή current, ενημερώνεται με τον γονέα της τρέχουσας κατάστασης που αποθηκεύεται στο πεδίο parent. Αυτή η διαδικασία, συνεχίζεται μέχρι η current να γίνει null (μέχρι να φτάσει στην αρχική). Τελικά η λίστα path, επιστρέφεται με το πλήρες μονοπάτι, από την αρχική κατάσταση έως την τρέχουσα.

public int getCost(), getDepth(), getState(): Αυτές οι μέθοδοι, είναι accessor methods.

public int misplacedTilesHeuristic(): Η μέθοδος αυτή, υλοποιεί ένα από τα πιο απλές ευρετικές μεθόδους για το πρόβλημα του παζλ. Συγκεκριμένα, χρησιμοποιεί τον αριθμό των τοποθεσιών όπου οι πλακίδες δεν βρίσκονται στην σωστή θέση, σε σχέση με τον στόχο (goal state). Αρχικά, ορίζεται μία μεταβλητή misplacedTiles, για να καταμετρήσει τις λανθασμένα τοποθετημένες πλακίδες. Έπειτα, η μέθοδος εξετάζει κάθε θέση του πίνακα state και συγκρίνει την τιμή της κάθε θέσης με την αντίστοιχη θέση στον πίνακα goalState. Αν η τιμή δεν είναι μηδέν (δηλαδή υπάρχει πλακίδα) και δεν είναι ίδια με την αντίστοιχη τιμή στον στόχο, τότε αυξάνεται ο μετρητής misplacedTiles. Τέλος, η μέθοδος επιστρέφει τον αριθμό των λανθασμένα τοποθετημένων πλακιδίων ως ευρετικό για το κόστος που χρησιμοποιείται για την αξιολόγηση της κάθε κατάστασης του παζλ.

public boolean equals(): Η μέθοδος equals χρησιμοποιείται για να εξετάσει εάν_ένα αντικείμενο PuzzleState είναι ίσο με ένα άλλο αντικείμενο. Αρχικά, η μέθοδος ελέγχει

εάν το αντικείμενο this είναι ίδιο με το αντικείμενο obj, χρησιμοποιώντας την σύγκριση μέσω του τελεστή ==. Αν είναι, τότε επιστρέφει αμέσως true, καθώς αυτό σημαίνει ότι τα δύο αντικείμενα είναι ακριβώς το ίδιο. Στη συνέχεια, ελέγχει εάν το αντικείμενο obj είναι null ή αν δεν ανήκει στην ίδια κλάση με το αντικείμενο this, χρησιμοποιώντας τις μεθόδους getClass() και getClass(). Αν είναι έτσι, τότε επιστρέφει false, διότι δύο αντικείμενα που δεν ανήκουν στην ίδια κλάση δεν μπορούν να είναι ίσα. Τέλος, αν οι προϋποθέσεις των προηγούμενων ελέγχων δεν ισχύουν, τότε δημιουργείται ένα νέο αντικείμενο τύπου PuzzleState με τη χρήση του obj, και ελέγχεται η ισότητα των πινάκων κατάστασης state του this και του νέου αντικειμένου, χρησιμοποιώντας τη στατική μέθοδο Arrays.deepEquals. Εάν οι πίνακες κατάστασης είναι ίσοι, τότε η μέθοδος επιστρέφει true, διαφορετικά επιστρέφει false. Η μέθοδος είναι σημαντική για την επιβεβαίωση της ισότητας μεταξύ αντικειμένων PuzzleState, βασιζόμενη κυρίως στη σύγκριση των πινάκων κατάστασης.

public int hashCode(): Η μέθοδος hashCode χρησιμοποιείται για τη δημιουργία ενός μοναδικού αριθμητικού κωδικού (hash code) που αντιστοιχεί σε κάθε αντικείμενο της κλάσης PuzzleState. Στη συγκεκριμένη υλοποίηση, ο αριθμητικός κωδικός υπολογίζεται με τη χρήση της στατικής μεθόδου Arrays.deepHashCode, η οποία δημιουργεί έναν hash code βάσει των περιεχομένων του πίνακα state με βάθος. Ο hash code που δημιουργείται είναι αναγκαίος για την αποδοτική αποθήκευση και ανάκτηση αντικειμένων σε δομές δεδομένων που βασίζονται σε hash, όπως οι συλλογές HashMap.

Test.java:

Η μέθοδος main είναι η είσοδος στο πρόγραμμά μας και αρχικοποιεί την επίλυση ενός γρίφου παζλ. Αρχικά, δημιουργεί ένα αντικείμενο Scanner για να λάβει είσοδο από τον χρήστη μέσω του πληκτρολογίου. Στη συνέχεια, ζητά από τον χρήστη να εισαγάγει την αρχική κατάσταση του παζλ, γραμμή προς γραμμή, χρησιμοποιώντας κενό ως διαχωριστικό. Οι αριθμοί που εισάγονται αποθηκεύονται σε έναν πίνακα 3x3 που αντιπροσωπεύει την αρχική κατάσταση του παζλ. Στη συνέχεια, δημιουργείται ένα αντικείμενο της κλάσης PuzzleSolver που θα χρησιμοποιηθεί για την επίλυση του παζλ. Τρέχει δύο διαφορετικούς αλγορίθμους αναζήτησης λύσης, το Uniform Cost Search (UCS) και το A* Search. Κάθε αλγόριθμος λύσης καλείται με την αρχική κατάσταση του παζλ που έχει εισαχθεί από τον χρήστη. Η παράμετρος false περνιέται στην μέθοδο solve για την εκτέλεση του UCS, ενώ η τιμή true περνιέται για τον A* Search. Τέλος, οι λύσεις κάθε αλγορίθμου εμφανίζονται στην οθόνη.

Κατασκευή Παιγνίου

**Η παρούσα άσκηση υλοποιήθηκε σε Java, σε 2 κλάσεις.*

Το παιχνίδι αποτελείται από δύο παίκτες (MIN και MAX), όπου κάθε παίκτης εναλλάσεται και τοποθετεί ένα από τα τρία γράμματα (C, S, E) σε οποιαδήποτε κενή θέση ενός πλέγματος 3x3.

Το παιχνίδι ξεκινά από μία αρχική κατάσταση όπου το γράμμα S υπάρχει υποχρεωτικά είτε στην αριστερή είτε στην δεξιά θέση της μεσαίας γραμμής. Το παιχνίδι τελειώνει αν ένας παίκτης σχηματίσει μία από τις δύο τελικές τριάδες ('CSE' ή 'ESC') οριζόντια, κατακόρυφα ή διαγωνίως σε διαδοχικές θέσεις του πλέγματος ή αν το πλέγμα γεμίσει, χωρίς να σχηματιστεί καμία από τις επιθυμητές τριάδες, με αποτέλεσμα ισοπαλία.

Για την κατασκευή του προγράμματος, θα χρησιμοποιηθεί ο αλγόριθμος MINIMAX, για να επιλέξει την κίνηση που θα κάνει κάθε φορά ο παίκτης MAX, λαμβάνοντας υπόψη την τρέχουσα κατάσταση του παιχνιδιού.

Χρονοδιάγραμμα 2ης Εργαστηριακής Άσκησης

Έκδοση	Ημερομηνία	Progress
1.0	27/3/2024	Ανακοίνωση 2ης εργαστηριακής άσκησης
2.0	22/4 - 1/5	Υλοποίηση άσκησης
3.0	16/5/2024	Σχόλια στον κώδικα
4.0	17/5/2024	Υλοποίηση Report 2ης εργαστηριακής άσκησης

Ανάλυση Χαρακτηριστικών

Η εργασία υλοποιήθηκε σε ένα μηχάνημα (Lenovo Ideapad 3), με τα εξής χαρακτηριστικά πυρήνα:

- AMD Ryzen 7 (7730U)
- 8 CPU cores
- 16 Threads
- Boost Clock up to 4.5GHz
- Base Clock 2.0GHz
- CPU Socket FP6
- Intergraded Graphics (Radeon Graphics)

Επιπλέον, η εργασία υλοποιήθηκε σε **Windows 11 Home**, χρησιμοποιώντας το **Visual Studio**.

GridGame.java:

Στην παρούσα κλάση, έχουμε ορίσει τα πεδία:

private static final char EMPTY: Το πεδίο αυτό, αναπαριστά ένα κενό κελί στο ταμπλό του παιχνιδιού.

private static final int MAX: Το πεδίο αυτό, αναπαριστά την constant τιμή που έχει ο παίκτης MAX.

private static final int MIN: Το πεδίο αυτό, αναπαριστά την constant τιμή που έχει ο παίκτης MIN.

private char[][] board: Το πεδίο αυτό, αναπαριστά το ταμπλό του παιχνιδιού

private int currentPlayer: Το πεδίο αυτό, κρατάει το ποιος παίκτης έχει σειρά. Αρχικοποιείται στον παίκτη MAX.

private Map<String, Integer> memo: Το πεδίο αυτό, χρησιμοποιείται για μεμονωμένες αποθηκεύσεις, αποθηκεύοντας καταστάσεις του πίνακα ως συμβολοσειρά και τις αντίστοιχες βαθμολογίες τους. Βοηθά στο να αποφευχθούν επαναλαμβανόμενα βήματα στον αλγόριθμο MINIMAX.

Έπειτα, έχουμε υλοποιήσει τις εξής μεθόδους:

public static void main(): Η μέθοδος main αποτελεί το σημείο εκκίνησης του προγράμματος. Εδώ δημιουργείται μία νέα έκδοση της κλάσης GridGame με το όνομα game, και έπειτα καλείται η μέθοδος runGame() για να ξεκινήσει το παιχνίδι. Η runGame() είναι υπεύθυνη για τη διαχείριση του βρόγχου του παιχνιδιού και την αλληλεπίδραση με τον χρήστη.

private void bestMove(): Η μέθοδος αυτή, υλοποιεί τη λειτουργικότητα της εύρεσης της καλύτερης δυνατής κίνησης για τον παίκτη MAX χρησιμοποιώντας τον αλγόριθμο Minimax. Αρχικά, αρχικοποιεί μεταβλητές για την καλύτερη τιμή και την καλύτερη κίνηση. Στη συνέχεια, εξερευνά όλες τις δυνατές κινήσεις που μπορεί να κάνει ο παίκτης MAX, δηλαδή τις κενές θέσεις στον πίνακα. Για κάθε κενή θέση, δοκιμάζει όλα τα πιθανά γράμματα που μπορεί να τοποθετήσει ο παίκτης MAX (C, E, S). Σε κάθε επανάληψη, τοποθετεί ένα γράμμα στη θέση αυτή και υπολογίζει την αξία της κίνησης χρησιμοποιώντας τη μέθοδο minimax. Στη συνέχεια, αφαιρεί το γράμμα από τη θέση για να επαναχρησιμοποιήσει τη θέση για άλλες κινήσεις. Κατά τη διάρκεια αυτού του βήματος, η μέθοδος ανανεώνει την καλύτερη τιμή και την καλύτερη κίνηση ανάλογα με την αξία της κίνησης που επιστρέφει ο αλγόριθμος Minimax. Τέλος, εφαρμόζει την καλύτερη κίνηση στον πίνακα και την εκτυπώνει στην οθόνη για να ενημερώσει τον χρήστη για την κίνηση που έγινε από τον παίκτη MAX.

private int minimax(): Η μέθοδος αυτή, υλοποιεί τον αλγόριθμο Minimax για τη λήψη της βέλτιστης κίνησης σε ένα παιχνίδι. Η μέθοδος λαμβάνει ως ορίσματα το βάθος αναζήτησης και μια λογική μεταβλητή που υποδηλώνει εάν ο τρέχων παίκτης είναι ο Maximizing Player. Αρχικά, ελέγχει εάν η κατάσταση του πίνακα έχει ήδη αποθηκευτεί στη μνήμη cache. Αν ναι, επιστρέφει το αντίστοιχο αποτέλεσμα από την cache. Στη συνέχεια, ελέγχει εάν υπάρχει νίκη ή εάν ο πίνακας είναι γεμάτος. Αν ισχύει ένα από τα παραπάνω, επιστρέφει το σκορ ανάλογα με τον παίκτη που είναι σε σειρά να παίξει και το αποθηκεύει στην cache. Στη συνέχεια, επιλέγει την αρχική τιμή bestValue ανάλογα με τον τρέχοντα παίκτη. Έπειτα, ελέγχει κάθε δυνατή κίνηση που μπορεί να γίνει από τον παίκτη σε κενό κελί. Για κάθε κίνηση, καλεί αναδρομικά τον εαυτό της αυξάνοντας το βάθος αναζήτησης και εναλλάσσοντας τη σειρά του παίκτη. Αφού λάβει το αποτέλεσμα της αναδρομικής κλήσης, επιστρέφει τη βέλτιστη τιμή για τον τρέχοντα παίκτη, είτε τη μέγιστη εάν είναι ο Maximizing Player είτε την ελάχιστη εάν είναι ο Minimizing Player. Τέλος, αποθηκεύει την καλύτερη τιμή για την τρέχουσα κατάσταση του πίνακα στην cache και την επιστρέφει. Με αυτόν τον τρόπο, η μέθοδος εξασφαλίζει ότι ο αλγόριθμος Minimax θα εξερευνήσει το δέντρο των κινήσεων με βάθος, αξιολογώντας και αποθηκεύοντας τα αποτελέσματα για μελλοντική χρήση προκειμένου να βρει τη βέλτιστη κίνηση.

private void runGame(): Η μέθοδος αυτή, υλοποιεί τον κύριο κύκλο του παιχνιδιού. Αρχικά, αρχικοποιεί τον πίνακα του παιχνιδιού και εκτυπώνει την αρχική του κατάσταση. Στη συνέχεια, εισέρχεται σε έναν ατέρμονα βρόγχο while, ο οποίος εκτελείται μέχρι να υπάρξει νικητής ή ισοπαλία. Κατά τη διάρκεια κάθε επανάληψης του βρόγχου, ελέγχεται ποιος παίκτης πρέπει να παίξει. Αν ο τρέχων παίκτης είναι ο MAX, καλείται η μέθοδος bestMove() για να επιλεχθεί η καλύτερη δυνατή κίνηση. Σε διαφορετική περίπτωση, εκτυπώνεται μήνυμα που υποδεικνύει τη σειρά του παίκτη MIN και ζητείται από τον χρήστη να εισάγει τη σειρά, τη στήλη και το γράμμα που επιθυμεί να τοποθετήσει στον πίνακα. Μετά τον έλεγχο εγκυρότητας της κίνησης, αν το πλακίδιο μπορεί να τοποθετηθεί στη θέση που επέλεξε ο παίκτης MIN, τοποθετείται στον πίνακα. Διαφορετικά, εκτυπώνεται μήνυμα σφάλματος και η επανάληψη συνεχίζεται. Στη συνέχεια, εκτυπώνεται ο πίνακας μετά την κίνηση. Στη συνέχεια, ελέγχεται αν υπάρχει νικητής ή αν ο πίνακας είναι γεμάτος. Αν κάποια από τις δύο συνθήκες ικανοποιείται, εκτυπώνονται τα αντίστοιχα μηνύματα νίκης ή ισοπαλίας, και ο βρόγχος τερματίζεται. Τέλος, η σειρά του παίκτη αλλάζει, προετοιμάζοντας τον επόμενο γύρο. Όταν ο βρόγχος τερματίζει, η μέθοδος κλείνει τον scanner που χρησιμοποιείται για είσοδο από το χρήστη.

private String boardToString(): Η μέθοδος αυτή, δημιουργεί μια αναπαράσταση του τρέχοντος πίνακα του παιχνιδιού ως αλφαριθμητικό. Χρησιμοποιεί ένα αντικείμενο StringBuilder για να δημιουργήσει το αλφαριθμητικό σταδιακά. Η μέθοδος επαναλαμβάνει κάθε γραμμή του πίνακα και για κάθε κελί στη γραμμή, προσθέτει την

τιμή του στο StringBuilder. Τέλος, επιστρέφει το αλφαριθμητικό που προέκυψε. Με αυτόν τον τρόπο, η μέθοδος μετατρέπει τον πίνακα του παιχνιδιού σε ένα αλφαριθμητικό που μπορεί να χρησιμοποιηθεί για διάφορους σκοπούς, όπως η αποθήκευση της κατάστασης του παιχνιδιού ή η χρήση ως κλειδί για την μνήμη cache στον αλγόριθμο Minimax.

private void initializeBoard(): Η μέθοδος αυτή, αρχικοποιεί τον πίνακα του παιχνιδιού με μια τυχαία αρχική κατάσταση. Χρησιμοποιεί ένα αντικείμενο της κλάσης Random για τη δημιουργία τυχαίου αριθμού ανάμεσα στο 0 και το 1. Ανάλογα με τον τυχαίο αριθμό που προκύπτει, επιλέγει αν θα τοποθετήσει το γράμμα 'S' σε μια από δύο προκαθορισμένες θέσεις στον πίνακα. Αυτό δημιουργεί διαφορετικές αρχικές συνθήκες για το παιχνίδι, προσθέτοντας ποικιλία και ενδιαφέρον στο παιχνίδι κάθε φορά που ξεκινάει. Η μέθοδος αυτή εκτελείται μία φορά στην αρχή του παιχνιδιού για να θέσει την αρχική κατάσταση του πίνακα.

private boolean canPlace(): Η μέθοδος αυτή, ελέγχει εάν μπορεί να τοποθετηθεί ένα γράμμα σε μια συγκεκριμένη θέση του πίνακα του παιχνιδιού. Πιο συγκεκριμένα, ελέγχει αν οι συντεταγμένες (row, col) που δίνονται είναι εντός των ορίων του πίνακα (δηλαδή, αν είναι μεταξύ 0 και 2 για έναν πίνακα 3x3) και εάν η συγκεκριμένη θέση είναι κενή, δηλαδή αν δεν έχει ήδη τοποθετηθεί κάποιο γράμμα σε αυτήν τη θέση. Η μέθοδος επιστρέφει true εάν η κίνηση είναι έγκυρη και μπορεί να πραγματοποιηθεί, και false διαφορετικά. Αυτή η λειτουργία είναι σημαντική για τον έλεγχο της εγκυρότητας των κινήσεων που κάνει ο παίκτης στο παιχνίδι, διασφαλίζοντας ότι οι κινήσεις γίνονται μόνο σε έγκυρες θέσεις του πίνακα και ότι δεν αντικαθιστούν ήδη υπάρχοντα γράμματα.

private boolean checkWin(): Η μέθοδος αυτή, ελέγχει εάν υπάρχει νικητής στο παιχνίδι. Συγκεκριμένα, ελέγχει αν υπάρχει κάποια γραμμή, στήλη ή διαγώνιος στον πίνακα του παιχνιδιού που αποτελείται από τα γράμματα "C", "S" και "E" σε οποιαδήποτε σειρά. Για να επιτευχθεί αυτό, η μέθοδος καλεί την επικεφαλίδα checkLines με προκαθορισμένα πρότυπα που αντιπροσωπεύουν τους διάφορους τρόπους που μπορεί να σχηματιστεί μια γραμμή που οδηγεί σε νίκη, όπως "CSE" και "ESC". Εάν οποιαδήποτε από αυτές οι συνθήκες πληρούνται, η μέθοδος επιστρέφει true, υποδηλώνοντας ότι υπάρχει νικητής στο παιχνίδι. Αν καμία από τις παραπάνω συνθήκες δεν πληρούνται, τότε η μέθοδος επιστρέφει false, υποδηλώνοντας ότι δεν υπάρχει ακόμα νικητής. Αυτή η λειτουργία είναι κρίσιμη για τον έλεγχο του παιχνιδιού και τον προσδιορισμό του νικητή.

private boolean checkLines(): Η μέθοδος αυτή, αναλαμβάνει τον έλεγχο για το εάν υπάρχει κάποια γραμμή ή διαγώνιος στον πίνακα του παιχνιδιού που περιέχει μια από τις προκαθορισμένες ακολουθίες γραμμάτων, που ορίζονται από τα μοτίβα που περνάμε στη μέθοδο. Η μέθοδος παίρνει έναν πίνακα patterns ως είσοδο, ο οποίος περιέχει τα μοτίβα που πρέπει να ελεγχθούν για νίκη. Κατά την εκτέλεσή της, η

μέθοδος δημιουργεί δύο συμβολοσειρές (rowString και colString) για κάθε σειρά και στήλη του πίνακα αντίστοιχα, συγκεντρώνοντας τα γράμματα που βρίσκονται σε κάθε γραμμή και στήλη. Στη συνέχεια, ελέγχει εάν οποιαδήποτε από αυτές οι συμβολοσειρές περιέχει ένα από τα μοτίβα που ορίζονται στον πίνακα patterns. Αν ναι, τότε επιστρέφει true, υποδηλώνοντας ότι υπάρχει νικητής. Στη συνέχεια, η μέθοδος ελέγχει τις δύο διαγωνίους του πίνακα για τα ίδια μοτίβα. Αν εντοπίσει οποιαδήποτε από τις δύο διαγώνιους να περιέχει ένα από τα μοτίβα, επιστρέφει true. Αν κανένα από τα παραπάνω δεν ισχύει, η μέθοδος επιστρέφει false, υποδηλώνοντας ότι δεν υπάρχει νικητής στον πίνακα του παιχνιδιού. Η λειτουργία αυτή είναι καθοριστική για τον έλεγχο της κατάστασης του παιχνιδιού και την αναγνώριση του νικητή.

private boolean isBoardFull(): Η μέθοδος αυτή, ελέγχει εάν ο πίνακας του παιχνιδιού είναι γεμάτος, δηλαδή αν όλα τα κελιά του πίνακα έχουν τιμή διαφορετική από την συμβολοσειρά EMPTY. Κατά την εκτέλεση, η μέθοδος επαναλαμβάνει όλα τα κελιά του πίνακα, χρησιμοποιώντας δύο εμβόλιμες μεταβλητές i και j, οι οποίες εκτελούνται από το 0 έως το 2 (ο πίνακας έχει μέγεθος 3x3). Κάθε φορά που ελέγχει ένα κελί, η μέθοδος ελέγχει αν η τιμή του είναι ίση με τη συμβολοσειρά EMPTY. Εάν βρει έστω και ένα κενό κελί, τότε η μέθοδος επιστρέφει false, υποδηλώνοντας ότι ο πίνακας δεν είναι γεμάτος. Αν όλα τα κελιά έχουν τιμή διαφορετική από το EMPTY, τότε η μέθοδος επιστρέφει true, υποδηλώνοντας ότι ο πίνακας είναι γεμάτος και δεν υπάρχουν άλλες ελεύθερες θέσεις για να τοποθετηθούν γράμματα. Αυτή η λειτουργία είναι κρίσιμη για τον έλεγχο του παιχνιδιού και την αναγνώριση εάν ο πίνακας έχει γεμάτο, προκειμένου να καθοριστεί αν έχει πραγματοποιηθεί ισοπαλία.

private void printBoard(): Η μέθοδος αυτή, είναι υπεύθυνη για την εκτύπωση του πίνακα του παιχνιδιού στην οθόνη. Κατά την εκτέλεσή της, η μέθοδος εμφανίζει έναν αναγνωριστικό "board" που υποδεικνύει την αρχή του πίνακα. Έπειτα, χρησιμοποιεί δύο εμβόλιμες μεταβλητές i και j για να προσπελάσει κάθε κελί του πίνακα. Για κάθε κελί, η μέθοδος ελέγχει εάν είναι κενό (δηλαδή αν έχει την τιμή EMPTY) και εκτυπώνει το σύμβολο " αντί για την τιμή του κελιού. Αν το κελί δεν είναι κενό, εκτυπώνει την τιμή του κελιού. Η μέθοδος συνεχίζει με αυτό τον τρόπο μέχρι να εκτυπωθούν όλες οι γραμμές του πίνακα. Με αυτόν τον τρόπο, η μέθοδος εξασφαλίζει ότι ο πίνακας του παιχνιδιού εμφανίζεται στην οθόνη με σαφήνεια και κάθε κενό κελί αναπαρίσταται με το σύμβολο ".

Move.java:

Στην παρούσα κλάση, έχουμε:

Η κλάση αυτή, περιέχει τον constructor Move. Αντιπροσωπεύει μια κίνηση σε ένα παιχνίδι και περιλαμβάνει τρεις μεταβλητές μέλη: row, col και letter (οι οποίες αντιπροσωπεύουν τη γραμμή, τη στήλη και το γράμμα που τοποθετείται σε ένα κελί του πίνακα του παιχνιδιού αντίστοιχα). Ο constructor, δέχεται τις τιμές αυτών των μεταβλητών ως ορίσματα και τις αναθέτει στα αντίστοιχα πεδία της κλάσης. Χρησιμοποιώντας αυτήν την κλάση, μπορούμε να αποθηκεύσουμε τις κινήσεις που κάνει κάθε παίκτης κατά τη διάρκεια του παιχνιδιού. Αυτό επιτρέπει την καταγραφή των ενεργειών των παικτών και τη διαχείριση του παιχνιδιού με βάση τις κινήσεις που έχουν γίνει. Επίσης, η χρήση αυτής της κλάσης καθιστά ευκολότερη τη μεταφορά και την επεξεργασία των δεδομένων που σχετίζονται με τις κινήσεις σε οποιοδήποτε σημείο του προγράμματος.