Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Εργασία Γ'

Κίνηση Pacman με MinMax

Δομές Δεδομένων

Αλέξανδρος Σαχίνης
Αλέξανδρος Τζήκας

[Δημιουργία Αλγορίθμου σε Java
Εύρεση Βέλτιστης Κίνησης Pacman
Χρήση Αλγορίθμου MinMax
Ανάλυση του Συνολικού Αλγορίθμου]

Περιγραφή του Προβλήματος

Σκοπός αυτής της εργασίας είναι η υλοποίηση της κίνησης του Pacman μέσα στο περιβάλλον του παιχνιδιού. Σε αυτή την εργασία στο περιβάλλον κίνησης των μονάδων έχουν προστεθεί και τέσσερις (4) τοίχοι-τετράγωνα μέσα από τους οποίους δεν μπορεί να περάσει κανένας χαρακτήρας του παιχνιδιού. Επίσης, έχουν δημιουργηθεί και τέσσερα (4) ανοίγματα (ένα σε κάθε πλευρά του περιβάλλοντος) μέσα από τα οποία μπορεί να περάσει ο “Pacman” και να οδηγηθεί αυτόματα στην απέναντι πλευρά του ταμπλό. Τα φαντάσματα δεν μπορούν να τον ακολουθήσουν σε αυτή του την κίνηση. Ο “Pacman” κερδίζει εάν καταφέρει να συγκεντρώσει όλες τις σημαίες ή να ξεφύγει από τα φαντάσματα μέχρι να παρέλθει ο καθορισμένος αριθμός κινήσεων (1000). Τα φαντάσματα έχουν ως στόχο τους να πιάσουν τον “Pacman” και για αυτόν τον λόγο έχουν υλοποιηθεί έτσι ώστε να κινούνται με κάποια στρατηγική.

Ο στόχος λοιπόν της εργασίας είναι η δημιουργία μιας δενδρικής δομής, η οποία θα αποθηκεύει τις πιθανές κινήσεις του “Pacman” και για κάθε κίνηση του “Pacman” τους αντίστοιχους δυνατούς συνδυασμούς κινήσεων των τεσσάρων (4) φαντασμάτων. Ουσιαστικά για κάθε κίνηση του Pacman” υπάρχουν το πολύ 256 συνδυασμοί κινήσεων των φαντασμάτων. Θεωρούμε ζευγάρι κινήσεων μία κίνηση του “Pacman” και έναν συνδυασμό κινήσεων των φαντασμάτων. Δεχόμενοι ότι για κάθε κίνηση του “Pacman” τα φαντάσματα θα διαλέξουν να κινηθούν με τέτοιον τρόπο έτσι ώστε να ελαχιστοποιήσουν την πιθανότητα νίκης του “Pacman”, θα χρησιμοποιήσουμε τον αλγόριθμο MinMax για να εντοπίσουμε την πιο αποδοτική κίνηση του “Pacman” σε βάθος 2 κινήσεων. Αυτό θα γίνει εκτιμώντας την αποδοτικότητα του κάθε ζευγαριού κινήσεων (δηλαδή την αποδοτικότητα του ζευγαριού κινήσεων “Pacman” – φαντασμάτων που οδηγεί σε φύλλο του δέντρου). Δηλαδή, αρχικά επιλέγεται ως κίνηση φαντασμάτων για κάθε κίνηση του “Pacman” η κίνηση αυτή που ελαχιστοποεί την συνάρτηση εκτίμησης και ύστερα αυτή η κίνηση των φαντασμάτων και αυτή η βαθμολογία χαρακτηρίζουν την εκάστοτε κίνηση του “Pacman”. Ως επόμενη κίνηση του “Pacman” διαλέγεται η κίνηση αυτή με μέγιστη βαθμολγία χαρακτηρισμού.

Επομένως, πριν από κάθε κίνηση ο “Pacman” αξιολογεί την κάθε δυνατή του κίνηση και τις αντίστοιχες δυνατές κινήσεις των φαντασμάτων και επιλέγει την κίνηση αυτή που μειώνει την πιθανότητα ήττας και αυξάνει την πιθανότητα νίκης.

Περιγραφή και Ανάλυση Αλγορίθμων και Διαδικασιών

- Η κλάση `Node89068978`

Πρώτο βήμα στην υλοποίηση του τελικού αλγορίθμου είναι η συμπλήρωση της κλάσης `Node89068978`. Η κλάση αυτή αναπαριστά έναν κόμβο στο δέντρο. Οι μεταβλητές `nodeX, nodeY` περιέχουν τις συντεταγμένες του κόμβου και η μεταβλητή `nodeMove` την κίνηση που αυτός αντιπροσωπεύει. Η μεταβλητή `nodeEvaluation` περιέχει την εκτίμηση αποδοτικότητας της συγκεκριμένης κίνησης με βάση τα κριτήρια που έχουμε θέσει. Ο δισδιάστατος πίνακας `currentGhostPos` περιέχει τις συντεταγμένες κάθε φαντάσματος στο περιβάλλον πριν ο “Pacman” κάνει την κίνησή του. Ομοίως, ο δισδιάστατος πίνακας `flagPos` περιέχει τις συντεταγμένες των σημαιών για αυτό το παιχνίδι. Προκειμένου να γνωρίζουμε την κατάσταση (πιασμένη ή όχι) κάθε σημαίας ορίζεται ο πίνακας `currentFlagStatus` που περιέχει την κατάσταση κάθε σημαίας σε μορφή `true, false`.

Θεωρήθηκε καλό να αποθηκευτεί σε δισδιάστατο πίνακα τύπου `Room` το περιβάλλον παιχνιδιού, για αυτό και ορίστηκε η μεταβλητή `Maze`.

Επιπρόσθετα, έχουν οριστεί οι παρακάτω μεταβλητές για κάθε κόμβο, οι οποίες θα φανούν χρήσιμες στην δημιουργία της δενδρικής δομής:

`int depth:` Δηλώνει το βάθος της κίνησης στο δέντρο που έχουμε δημιουργήσει

`Node89068978 parent:` Δηλώνει τον κόμβο που αποτελεί τον πατέρα του κόμβου στον οποίο βρισκόμαστε

`ArrayList<Node89068978> children:` Μια δομή αποθήκευσης που περιλαμβάνει όλα τα παιδιά του κόμβου

`public boolean leaf:` Δηλώνει εάν ο συγκεκριμένος κόμβος είναι φύλλο ή όχι του δέντρου

Τώρα ως προς τις υλοποιημένες συναρτήσεις:

- Ο “Constructor” `Node89068978(int dep, Node89068978 par, int x, int y, int move, Room[][] Space)`

  Λαμβάνει ως ορίσματα τέσσερις ακεραίους:

  `x:` κάθετη συντεταγμένη του κόμβου

  `y`: οριζόντια συντεταγμένη του κόμβου

  `move`: η κίνηση του “Pacman”, την οποία αντιπροσωπεύει ο κόμβος

  `dep:` το βάθος του κόμβου στο δέντρο

  έναν δισδιάστατο πίνακα τύπου `Room` που θα περιέχει το περιβάλλον παιχνιδιού και έναν κόμβο που αποτελεί τον πατέρα του κόμβου αυτού.

  Η συνάρτηση απλά αντιστοιχεί τις τιμές των ορισμάτων στις κατάλληλες μεταβλητές της κλάσης.

Οι βασικές συναρτήσεις που εντοπίζουν τις μονάδες και τις καταστάσεις αυτών στο περιβάλλον παιχνιδιού είναι οι εξής:

- `public int[][] findGhosts()`

  Η συνάρτηση αυτή ελέγχει κάθε στοιχείο τύπου `Room` του `Maze` έτσι ώστε να εντοπίσει τα κελιά εκείνα που περιέχουν φαντάσματα. Ακολουθώντας τη σειρά εντοπισμού των φαντασμάτων εισάγει τις συντεταγμένες τους σε έναν δισδιάστατο πίνακα, ο οποίος στο τέλος θα περιέχει την θέση όλων των φαντασμάτων στο `Maze`. Αυτός ο πίνακας εκχωρείται στην `currentGhostPos` μεταβλητή της κλάσης.

- `private int[][] findFlags()`

  Ακριβώς η ίδια λογική της παραπάνω συνάρτησης εφαρμόζεται έτσι ώστε να βρεθούν οι συντεταγμένες όλων των σημαιών του περιβάλλοντος και να

αποθηκευτούν οι θέσεις τους σε έναν δισδιάστατο πίνακα, ο οποίος ανά γραμμή θα περιέχει τις συντεταγμένες μιας εκ των σημαιών. Αυτός ο πίνακας εκχωρείται στην `flagPos` μεταβλητή της κλάσης.

- `private boolean[] checkFlags()`

  Για κάθε σημαία (που καταλαμβάνει την θέση i του πίνακα `flagPos`) γίνεται έλεγχος της σημαίας για να βρεθεί εάν έχει ήδη κατακτηθεί από τον “Pacman”. Σε αυτήν την περίπτωση αποθηκεύεται σε έναν πίνακα `boolean` στην θέση i η τιμή `true`. Σε άλλη περίπτωση αποθηκεύεται στην θέση αυτή η τιμή `false`. Με αυτόν τον τρόπο αναγνωρίζονται οι σημαίες ως “captured” ή “uncaptured”.

Παρακάτω παρατίθεται η εξήγηση των βοηθητικών συναρτήσεων που υλοποιήθηκαν:

- `private int pointToPointDistance(int[] coord, int x, int y)`

  Λαμβάνοντας τις συντεταγμένες του πρώτου σημείου σε μορφή πίνακα και τις συντεταγμένες του δεύτερου σημείου ως ακεραίους υπολογίζει την απόσταση Μανχάταν μεταξύ των δύο σημείων.

- `private int[] pointToEveryFlagDistance(int x, int y)`

  Επιστρέφει έναν πίνακα που στο i-στό στοιχείο του περιέχει την απόσταση Μανχάταν μεταξύ του σημείου `(x, y)` και της i-στής σημαίας του πίνακα `flagPos`.

- `private int[] densityAroundFlags()`

  Επιστρέφει το αντίστροφο της μέσης απόστασης των φαντασμάτων γύρω από κάθε σημαία σε μορφή πίνακα. Δηλαδή το i-στό στοιχείο του πίνακα επιστροφής περιέχει το αντίστροφο άθροισμα των αποστάσεων Μανχάταν όλων των φαντασμάτων από την i-στή σημαία του πίνακα `flagPos` διαιρεμένο με τον αριθμό των φαντασμάτων. Όσο πιο μικρή η τιμή του στοιχείου i, τόσο πιο πολύ απέχουν (κατά μέσο όρο) τα φαντάσματα από την i-στή σημαία.

- `private int bestFlag()`

  Ουσιαστικά εντοπίζει την καλύτερη-πιο αποδοτική σημαία προς την οποία θα έπρεπε να κινηθεί ο “Pacman” συνδυάζοντας 2 κριτήρια:

  1) Την πυκνότητα φαντασμάτων γύρω από την κάθε σημαία (όσο μικρότερη είναι αυτή η τιμή τόσο καλύτερο είναι να κυνηγήσει αυτήν την σημαία, αφού η πυκνότητα είναι αντιστρόφως ανάλογη της απόστασης των φαντασμάτων από την σημαία). Η τιμή αυτή βρίσκεται για κάθε σημαία μέσω της `densityAroundFlags()`.

  2) Την απόσταση του “Pacman” από την σημαία αυτή (όσο μεγαλύτερη αυτή η τιμή τόσο χειρότερη η κίνηση). Βρίσκεται η απόσταση της θέσης του “Pacman” πριν κάνει κάποια κίνηση σε αυτόν τον γύρο από όλες τις σημαίες μέσω της `pointToEveryFlagDistance(int x, int y)`.

Τα δύο αυτά κριτήρια-τιμές συνδυάζονται με κάποια συγκεκριμένα βάρη, τα οποία έχουν βρεθεί πειραματικά και με χρήση της συνάρτησης 1/x και προκύπτει η εκτίμηση για κάθε σημαία, η οποία εξηγείται ποιοτικά ως εξής:

> Όσο μεγαλώνει η απόσταση από την σημαία μειώνεται η τιμή που προστίθεται στην εκτίμηση (αυτό δείχνει ότι η κίνηση δεν είναι και τόσο καλή), ενώ όσο μειώνεται η πυκνότητα φαντασμάτων (μειώνεται η έξοδος της `densityAroundFlags`) γύρω από την σημαία μειώνεται η τιμή που αφαιρείται από την εκτίμηση (αυτό δείχνει ότι η σημαία αυτή είναι καλή επιλογή κίνησης).

Τέλος, αφού γίνει η εκτίμηση της κάθε σημαίας επιλέγεται η σημαία με την καλύτερη βαθμολογία η οποία όμως δεν έχει ήδη πιαστεί, δηλαδή η σημαία i για την οποία το `flagEvaluate[i]` είναι η μέγιστη τιμή του πίνακα. Αυτή θα είναι η σημαία προς την οποία θα πρέπει να κινηθεί ο “Pacman” σε αυτόν τον γύρο. Στην συνάρτηση `evaluate()` η αξιολόγηση της πιθανής κίνησης `nodeMove` θα ελέγχει εάν η τελευταία οδηγεί ή όχι τον “Pacman” στην σημαία που προκύπτει από την `bestFlag()`.

- `private int pointToClosestGhostDistance()`

  Υπολογίζει την απόσταση του πιο κοντινού φαντάσματος από την θέση του “Pacman” μετά την υποθετική κίνηση `nodeMove`. Ως κριτήριο για την επιλογή της επόμενης κίνησης του “Pacman” σε αυτόν τον γύρο είναι η επιλεγμένη κίνηση να μη φέρνει τα φαντάσματα κοντά στον “Pacman”. Ως κριτήριο της πιθανής κίνησης στην `evaluate()` λοιπόν είναι η μικρότερη απόσταση από φάντασμα μετά την κίνηση (όσο μικρότερη η απόσταση αυτή, τόσο μικραίνει η βαθμολογία της πιθανής κίνησης).

- `private Boolean possibleNextGhostPosition()`

  Ελέγχει εάν η θέση του “Pacman” μετά την κίνηση `nodeMove` είναι θέση στην οποία μπορεί να καταλήξει φάντασμα στον επόμενο γύρο κινήσεων (απέχει δηλαδή ένα κουτάκι από φάντασμα, άρα απόσταση Μανχάταν από αυτό 1 και δεν είναι σημαία, αφού τα φαντάσματα δεν μπορούν να καταλήξουν σε σημαία). Μια τέτοια κίνηση είναι σχεδόν απαγορευτική για τον “Pacman” διότι αυξάνεται πολύ η πιθανότητα να πιαστεί από φάντασμα.

- `private int distToEdge()`

  Αυτή η συνάρτηση υπολογίζει την μικρότερη απόσταση του “Pacman” στην θέση `(nodeX, nodeY)` από τοίχο στα περιμετρικά του περιβάλλοντος. Είναι ένα μέτρο της απόστασης του “Pacman” από τα άκρα της πίστας. Στόχος του “Pacman” πρέπει να είναι να φτάνει κοντά σε τοίχους διότι μπορεί να ξεφύγει από τα φαντάσματα μέσω των ανοιγμάτων που υπάρχουν σε αυτούς.

- `public boolean isleaf(boolean isOnFlag)`

  Ελέγχει εάν ο συγκεκριμένος κόμβος είναι φύλλο και αποθηκεύει το αποτέλεσμα στην μεταβλητή `leaf`. Ως φύλλα θεωρούμε τους τερματικούς κόμβους του δέντρου. Φτάνοντας σε αυτούς είτε τελειώνει το παιχνίδι, είτε αντιπροσωπέυουν την καλύτερη δυνατή κίνηση και δεν χρειάζεται να προχωρήσουμε βαθύτερα. Καλύτερη κίνηση είναι αυτή που μας οδηγεί στην κατάκτηση μιας σημαίας. Η κίνηση αυτή θέλουμε να είναι η επόμενη, ανεξάρτητα από την θέση των φαντασμάτων. Κίνηση που τελειώνει το παιχνίδι είναι αυτή που μας δίνει nodeEvaluation=-100 και που ταυτόχρονα δεν προέρχεται από κόμβο που παριστά σημαία. Αν προέρχεται από κόμβο-σημαία (δηλαδή η προηγούμενη θέση ήταν πάνω σε σημαία - ένα node δηλώνει την θέση μετά την κλινηση), τότε ένα φάντασμα που βρίσκεται δίπλα στον Pacman δε θα μπορέσει να κινηθεί προς τη σημαία (δηλαδή να ανταλλάξει θέση με αυτόν), ούτε να παραμείνει ακίνητο. Συνεπώς θα απομακρυνθεί από τον Pacman. Έτσι, υπάρχει μια αρκετά καλή πιθανότητα ο Pacman να γλυτώσει. Για να είμαστε σίγουροι, συνεχίζουμε την ανάπτυξη του δένδρου ώστε να δούμε και τις κινήσεις των υπολοίπων φαντασμάτων, από τις οποίες τελικά θα καθοριστεί το αν τελικά ο Pacman ξεφεύγει ή όχι.

- `public int getMove()`

  Επιστρέφει την μεταβλητή `nodeMove` της `Node89068978`.

- `public int getDepth()`

  Επιστρέφει την μεταβλητή `depth` της `Node89068978`.

- `public double getEval()`

  Επιστρέφει την μεταβλητή `nodeEvaluation` της `Node89068978`.

- `public ArrayList<Node89068978> getChildren()`

  Επιστρέφει την μεταβλητή `children` της `Node89068978`

Τέλος, εξηγείται η σημαντικότερη συνάρτηση της κλάσης `Node89068978`:

- `public double evaluate()`

  Η συγκεκριμένη συνάρτηση ελέγχει αρχικά αν ο “Pacman” οδηγείται σε κίνηση που σίγουρα θα το οδηγήσει σε ήττα οπότε και την αξιολογεί με -100. Ως τέτοιες κινήσεις λογίζονται αυτές που ανήκουν στην παρακάτω κατηγορία:

  - Κίνηση που καταλήγει σε θέση που ήδη υπάρχει φάντασμα.

  Στη συνέχεια, ελέγχεται το αν η συγκεκριμένη κίνηση οδηγεί σε σημαία η οποία δεν έχει κατακτηθεί ακόμη από τον “Pacman”. Τότε, η αξιολόγηση γίνεται η μέγιστη (100).

  Τέλος, για τις υπόλοιπες, μη ακραίες περιπτώσεις, ακολουθείται μια γενική μέθοδος αξιολόγησης:

Πρώτα ελέγχεται το αν η συγκεκριμένη κίνηση αποτελεί πιθανή κίνηση κατάληξης φαντάσματος. Αν αυτό συμβαίνει, αφαιρούνται 500 μονάδες για να αποτρέψουν τον “Pacman” να κινηθεί προς τα εκεί. Σε διαφορετική περίπτωση δεν αφαιρούνται μονάδες και προχωράμε στον αριθμητικό υπολογισμό της `evaluation()` ο οποίος βασίζεται σε 3 κριτήρια για τον κόμβο:

- Η απόσταση από το κοντινότερο φάντασμα:

  Όσο πιο μακριά βρίσκεται από το κοντινότερο φάντασμα, τόσο μικρότερη τιμή αφαιρείται από την αξιολόγηση. Η αλλαγή δε γίνεται γραμμικά, αλλά χρησιμοποιώντας τη συνάρτηση 1/x

- Η απόσταση από την καλύτερη σημαία:

  Όσο πιο μακριά βρίσκεται από αυτή, τόσο μικραίνει το ποσό που προστίθεται στην αξιολόγηση. Η πρόσθεση αυτή επίσης δε γίνεται γραμμικά, αλλά ακολουθώντας τη συνάρτηση 1/x

- Η απόσταση από τον κοντινότερο τοίχο:

  Ο “Pacman” επιβραβεύεται όταν κινείται κοντά στον τοίχο, γιατί έτσι μπορεί να ξεφεύγει από τα ανοίγματα

Στα κριτήρια αυτά έχουν δοθεί κάποια επιπλέον βάρη, ώστε να δώσουμε προτεραιότητα σε αυτά που θεωρούμε σημαντικότερα. Το πιο σημαντικό κριτήριο στο δικό μας αλγόριθμο, είναι να πλησιάζει ο “Pacman” την καλύτερη σημαία, ώστε να μπορεί να κερδίσει κατακτώντας και τις 4. Παράλληλα, προσπαθεί να αποφεύγει τα φάντασματα, επιλέγοντας κινήσεις που το απομακρύνουν από το κοντινότερο του. Τέλος, ενθαρρύνεται να κινείται στους τοίχους που βρίσκονται περιμετρικά.

- Η κλάση `Creature`

I. Δημιουργία Δενδρικής Δομής

  - Δημιουργία Τμήματος Δέντρου Προερχόμενο Από τις Δυνατές Κινήσεις του “Pacman”

    `void createSubTreePacman(int depth, Node89068978 parent, Room[][] Maze, int[] currPacmanPosition)`

Δημιουργείται μια μεταβλητή τύπου `Room[][]` με όνομα `MazeCopy` που θα περιέχει ένα αντίγραφο του περιβάλλοντος παιχνιδιού. Επίσης, χρησιμοποιείται μια μεταβλητή τύπου `boolean` που θα αναγνωρίζει εάν η παρούσα θέση του “Pacman” είναι πάνω σε σημαία.

Στην συνέχεια, για κάθε δυνατή κατεύθυνση κίνησης του Pacman, μια μεταβλητή `int[] nextPacmanPosition` αρχικοποιείται στην τιμή του πίνακα `int[] currPacmanPosition` και:

Εάν στην στην κατεύθυνση της εκάστοτε κίνησης δεν υπάρχει τοίχος συμβαίνουν τα εξής:

Αρχικά αυξάνεται ο αριθμός των nodes. Στην συνέχεια αντιγράφεται στο `MazeCopy` το `Maze`. Υπολογίζεται η νέα θέση του “Pacman” με βάση την συγκεκριμένη κίνηση και την αρχική του θέση και μετακινείται στην νέα αυτή θέση στο αντίγραφο του περιβάλλοντος παιχνδιού. Στο αντίγραφο αυτό λοιπόν βλέπουμε το ταμπλό μετά την εκάστοτε κίνηση του “Pacman”. Αυτή κατάσταση αποτελεί έναν νέο κόμβο του δέντρου στο πρώτο επίπεδο (δυνατές κινήσεις του “Pacman”). Προστίθεται λοιπόν ένας κόμβος ως παιδί της ρίζας (του parent) με τα χαρακτηριστικά του κόμβου αυτού, δηλαδή με πατέρα τον parent, θέση την νέα θέση του “Pacman”, κίνηση την εκάστοτε κίνηση, βάθος το βάθος του επιπέδου κίνησης στο οποίο βρισκόμαστε και περιβάλλον αυτό που προκύπτει μετά την κίνηση του “Pacman” και πριν την κίνηση των φαντασμάτων.

Στην περίπτωση που η κίνηση αυτή (μια εκ των τεσσάρων στην περίπτωση που όλες οι κινήσεις του “Pacman” είναι δυνατές) δεν αποτελεί «φύλλο» (δηλαδή τερματική κίνηση που οδηγεί στην κατάκτηση σημαίας ή σε σύγκρουση με φάντασμα) καλείται η συνάρτηση `createSubTreePacman` (για κάθε παιδί) για να δημιουργηθεί υποδέντρο με ρίζα την συγκεκριμένη κίνηση του “Pacman” και φύλλα τους δυνατούς συνδυασμούς κινήσεων των φαντασμάτων για κάθε δυνατή κίνηση του “Pacman”. Αυξάνεται και ο δείκτης παιδιών.

  - Δημιουργία Τμήματος Δέντρου Προερχόμενο Από τις Δυνατούς Συναδυασμούς Κινήσεων Φαντασμάτων για Κάθε Δυνατή Κίνηση του “Pacman”

    `void createSubTreeGhosts(int depth, Node89068978 parent, Room[][] Maze, int[] currPacmanPosition)`

Δημιουργείται μια μεταβλητή τύπου `Room[][]` με όνομα `MazeCopy` που θα περιέχει ένα αντίγραφο του περιβάλλοντος παιχνιδιού και μια μεταβλητή τύπου `ArrayList<int[][]> avPos`, η οπόια θα περιέχει μια συλλογή δισδιάστατων πινάκων κάθε ένας από τους οποίους θα περιέχει έναν από τους δυνατούς συνδυασμούς κινήσεων των τεσσάρων

φαντασμάτων. Στην μεταβλητή `currPacPos` ανατίθεται η θέση του “Pacman” μετά την εκάστοτε κίνησή του (στο πρώτο επίπεδο της δενδρικής δομής), αφού όταν θα καλείται η συνάρτηση `createSubTreeGhosts` θα έχει ως όρισμα το αντίγραφο του Maze με ανανεωμένη την θέση του «Pacman».

Έπειτα, για κάθε έναν από τους συνδυασμούς συμβαίνει το εξής:

Αντιγράφεται στο `MazeCopy` το `Maze`. Ανανεώνονται οι θέσεις των φαντασμάτων σε αυτό το ατίγραφο σύμφωνα με την κίνηση του καθενός στον εκάστοτε συνδυασμό κινήσεων.

Σαν παιδί στον πατέρα του υποδέντρου που περνιέται μέσω των ορισμάτων, τοποθετείται ο δημιουργημένος κόμβος με βάθος το βάθος ορίσματος, πατέρα αυτόν του ορίσματος, αντίγραφο περιβάλλοντος το ανανεωμένο μετά τις κινήσεις των φαντασμάτων και θέση του “Pacman” αυτή του ορίσματος.

II. Υπολογισμός της Επόμενης Κίνησης του “Pacman”

```
public int calculateNextPacmanPosition(Room[][] Maze, int[] currPosition)
```

Αρχικά τίθεται στο μηδέν ο συνολικός αριθμός κόμβων του δέντρου.

Δημιουργούνται δύο κόμβοι τύπου `Node89068978`. Ο ένας αποτελεί την ρίζα όλης της δενδρικής δομής και για αυτό ονομάζεται `root`. Ο κόμβος έχει βάθος 0, πατέρα κανέναν, θέση του “Pacman” την θέση του “Pacman” πριν αυτός κάνει κάποια κίνηση, κίνηση κάποια αδύνατη και με αντίγραφο περιβάλλοντος το αρχικό περιβάλλον πριν γίνει κάποια κίνηση.

Καλείται η συνάρτηση `createSubTreePacman` με όρισμα βάθους την μονάδα (βρισκόμαστε στο πρώτο επίπεδο του δέντρου), πατέρα των κόμβων που θα δημιουργηθούν την ρίζα, περιβάλλον παιχνιδιού το αρχικό και θέση του “Pacman” την αρχική του θέση.

Με αυτόν τον τρόπο θα δημιουργηθούν 4 παιδιά της ρίζας με βάθος κόμβου 1, πατέρα την ρίζα, θέση του “Pacman” αυτή που προκύπτει μετά από μια εκ των δυνατών κινήσεών του και περιβάλλον παιχνιδιού αυτό στο οποίο έχει πια ανανεωθεί η θέση του “Pacman”.

Για κάθε ένα από αυτά τα παιδιά (έαν αυτό δε αποτελεί «φύλλο») δημιουργείται ένα υποδέντρο – σύνολο παιδιών που εξομοιώνει τις δυνατές κινήσεις των φαντασμάτων στην συγκεκριμένη κίνηση του “Pacman”. Κάθε παιδί θα περιέχει και ένα μοναδικό περιβάλλον παιχνιδιού με μοναδικό συνδυασμό κινήσεων “Pacman”-φαντασμάτων.

Κάθε φύλλο του δευτέρου επιπέδου τώρα έχει μια τιμή αξιολόγησης. Κάθε ξεχωριστό τελικό ταμπλό παιχνιδιού λοιπόν αξιολογείται αφού αποτελεί μία εκ των δυνατών απεικονίσεων του ταμπλό σε βάθος 2 κινήσεων και με χρήση του αλγορίθμου MinMax επιλέγεται από τον “Pacman” η καλύτερη-βέλτιστη (σε βάθος 2) κίνηση, θεωρώντας ότι κάθε φορά τα φαντάσματα κινούνται έτσι ώστε να ελαχιστοποιήσουν την πιθανότητα νίκης του “Pacman”. Διαλέγουν για την εκάστοτε κίνηση του “Pacman” το τελικό ταμπλό με την μικρότερη τιμή αξιολόγησης και κινούνται ανάλογα. Επιλέγουν δηλαδή 4 συνολικά ταμπλό (ένα για κάθε δυνατή κίνηση του “Pacman”. Ο “Pacman” κινείται έτσι ώστε να μεγιστοποιήσει την πιθανότητα νίκης, διαλέγει δηλαδή από τις 4 επιλογές ταμπλό των φαντασμάτων αυτή με την μεγαλύτερη τιμή αξιολόγησης.

III. Εφαρμογή Αλγορίθμου MinMax

Η συνάρτηση `minimax` υλοποιείται καλώντας την `max` για την ρίζα του δέντρου.

a) `public static Node89068978 max(Node89068978 d, double a, double b)`

Η συνάρτηση αυτή δέχεται ως όρισμα έναν κόμβο και 2 τιμές . Ο κόμβος αυτός αποτελεί την ρίζα του υποδέντρου στο οποίο θα εφαρμόσουμε τον αλγόριθμο.

Εάν η ρίζα `d` αποτελεί «φύλλο» (τερματική κατάσταση) ή κόμβο βάθους ίσο με το μέγιστο βάθος στο οποίο επιθυμούμε να φτάσουμε επιστρέφεται αυτός ο κόμβος, αφού δεν έχει νόημα η βαθύτερη αναζήτηση.

Εάν η ρίζα `d` δεν ικανοποιεί τα παραπάνω κριτήρια είναι απαραίτητη η βαθύτερη αναζήτηση.Για αυτό εφαρμόζεται η συνάρτηση `min` σε κάθε ένα από τα παιδιά του κόμβου `d` (με ρίζα δηλαδή το εκάστοτε παιδί). Για κάθε παιδί λοιπόν, βρίσκεται το παιδί του το οποίο έχει την μικρότερη τιμή αξιολόγησης και επιστρέφεται. Τώρα από τα ελάχιστα που θα προκύψουν (το παιδί του κάθε παιδιού με ελάχιστη τιμή αξιολόγησης) επιλέγεται το μέγιστο και επιστρέφεται τελικά.

Για κάθε επανάληψη του βρόχου γίνεται και το εξής:

Αν η μέγιστη των ελαχίστων τιμών αξιολόγησης που έχουν προκύψει είναι μεγαλύτερη από την σταθερά a, αυτή ανανεώνεται αφού η μέγιστη τιμή αξιολόγησης που θα πάρουμε με την max είναι μεγαλύτερη από την a και το νόημα της a είναι να δίνει την μέγιστη δυνατή τιμή.

Αν η καλύτερη (ελάχιστη) τιμή του min σε ανώτερο επίπεδο στην παρούσα φάση είναι μικρότερη από το a τότε δεν έχει νόημα η περαιτέρω αναζήτηση σε αυτό το max (στο υποδέντρο με ρίζα d), αφού το a θα αλλάξει μόνο εάν πρόκειται να αυξηθεί ενώ ο από πάνω min θα διαλέξει την ελάχιστη δυνατή τιμή και αυτή σίγουρα δεν θα είναι αυτή που προέρχεται από αυτό το max, αφού θα προκύψει τιμή μεγαλύτερη από αυτή που τώρα έχει ο ανώτερος min σε αυτό το υποδέντρο.

b) `public static Node89068978 min(Node89068978 d, double a, double b)`

Η συνάρτηση αυτή δέχεται ως όρισμα έναν κόμβο και 2 σταθερές που είναι `static` στην κλάση. Ο κόμβος αυτός αποτελεί την ρίζα του υποδέντρου στο οποίο θα εφαρμόσουμε τον αλγόριθμο.

Εάν η ρίζα `d` αποτελεί «φύλλο» (τερματική κατάσταση) ή κόμβο βάθους ίσο με το μέγιστο βάθος στο οποίο επιθυμούμε να φτάσουμε επιστρέφεται αυτός ο κόμβος, αφού δεν έχει νόημα η βαθύτερη αναζήτηση.

Εάν η ρίζα `d` δεν ικανοποιεί τα παραπάνω κριτήρια είναι απαραίτητη η βαθύτερη αναζήτηση.Για αυτό εφαρμόζεται η συνάρτηση `max` σε κάθε ένα από τα παιδιά του κόμβου `d` (με ρίζα δηλαδή το εκάστοτε παιδί). Για κάθε παιδί λοιπόν, βρίσκεται το παιδί του το οποίο έχει την μεγαλύτερη τιμή αξιολόγησης και επιστρέφεται. Τώρα από τα μέγιστα που θα προκύψουν (το παιδί του κάθε παιδιού με μέγιστη τιμή αξιολόγησης) επιλέγεται το ελάχιστο και επιστρέφεται τελικά.

Για κάθε επανάληψη του βρόχου γίνεται και το εξής:

Αν η ελάχιστη των μεγίστων τιμών αξιολόγησης που έχουν προκύψει είναι μικρότερη από την σταθερά b, αυτή ανανεώνεται αφού η ελάχιστη τιμή αξιολόγησης που θα πάρουμε με την min είναι μικρότερη από την b και το νόημα της b είναι να δίνει την ελάχιστη δυνατή τιμή.

Αν η καλύτερη (μέγιστη) τιμή του max σε ανώτερο επίπεδο στην παρούσα φάση είναι μεγαλύτερη από το b τότε δεν έχει νόημα η περαιτέρω αναζήτηση σε αυτό το min (στο υποδέντρο με ρίζα d), αφού το b θα αλλάξει μόνο εάν πρόκειται να μειωθεί ενώ ο από πάνω max θα διαλέξει την μέγιστη δυνατή τιμή και αυτή σίγουρα δεν θα είναι αυτή που προέρχεται από αυτό το min, αφού θα προκύψει τιμή μικρότερη από αυτή που τώρα έχει ο ανώτερος max σε αυτό το υποδέντρο.

Ειδικά στην δική μας περίπτωση:

Καλείται η `minimax` με όρισμα την ρίζα του δέντρου.

Καλείται δηλαδή η max για την ρίζα του δέντρου. Επειδή η ρίζα δεν είναι φύλλο ή κόμβος μέγιστου βάθους καλείται η min για κάθε υποδέντρο (δηλαδή με όρισμα τα παιδιά της ρίζας).

Για κάθε παιδί επιπέδου 1 λοιπόν:

> Επειδή αυτό το παιδί δεν είναι φύλλο ή κόμβος μέγιστου βάθους καλείται η max για το κάθε παιδί τους (επιπέδου 2). Επειδή αυτά τα παιδιά είναι κόμβοι μεγίστου βάθους επιστρέφονται αυτούσια.
>
> Για κάθε παιδί επιπέδου 1 (μέσα στην min του) βρίσκεται το παιδί του με ελάχιστη τιμή αξιολόγησης και επιστρέφεται ως αποτέλεσμα της min του παιδιού επιπέδου 1.

Για την ρίζα (μέσα στην max της) βρίσκεται ο κόμβος με μέγιστη την τιμή αξιολόγησης από τους κόμβους που έχουν επιλεχθεί/επιστραφεί με το min των παιδιών της.

Η `calculateNextPacmanPosition` λοιπόν χρησιμοποιεί την `minimax` και βρίσκει το μοναδικό ταμπλό σε βάθος 2 που αποτελεί την επιθυμητή κατάσταση σε βάθος 2 κινήσεων και επιλέγει για την κίνηση του “Pacman” την κίνηση που περιέχεται σε αυτόν τον κόμβο.

Με αυτόν τον τρόπο ολοκληρώνεται η υλοποίηση της συγκεκριμένης εργασίας.

Συμπεράσματα

Είναι εμφανής η σημασία του αλγορίθμου MinMax σε τέτοια προβλήματα στα οποία επικρατεί αλληλεπίδραση μεταξύ των δύο πλευρών (η κίνηση του ενός επηρεάζει την μετέπειτα κίνηση του άλλου) αλλά και η συνεισφορά του αλγορίθμου α-β pruning γαι την μείωση των υπολογιστικών απαιτήσεων (χρονικά και χωρικά).

Για να εφαρμοστεί ο αλγόριθμος MinMax υποθέτουμε ότι τα φαντάσματα κάνουν πάντα την καλύτερη για εκείνα κίνηση.