Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Εργασία Β'

Κίνηση Pacman

Δομές Δεδομένων

Αλέξανδρος Σαχίνης
Αλέξανδρος Τζήκας

[Δημιουργία Αλγορίθμου σε Java
Εύρεση Βέλτιστης Κίνησης Pacman
Ανάλυση του 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) σημαίες. 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 αρχικά η εύρεση κάποιων αρκετά χαρακτηριστικών κριτηρίων με βάση τα οποία θα δημιουργείται μια εκτίμηση της αποδοτικότητας της κάθε δυνατής κίνησης του “Pacman”. Στην συνέχεια, εκτιμάται η αποδοτικότητα της κάθε δυνατής κίνησης και ως επόμενη κίνηση του “Pacman” επιλέγεται η βέλτιστη κίνηση, δηλαδή αυτή με την καλύτερη βαθμολογία στην εκτίμηση αποδοτικότητας.

Σε κάθε περίπτωση δηλαδή που έρχεται η σειρά του “Pacman” να κάνει κίνηση πρέπει να βρεθεί η κίνηση αυτή που τον οδηγεί μεν πιο κοντά σε κάποια μη-πιασμένη σημαία αλλά ταυτόχρονα τον διατηρεί σε μία καλή απόσταση από τα φαντάσματα, αλλά και από τα άκρα του περιβάλλοντος, έτσι ώστε να μειωθεί η πιθανότητα περικύκλωσης και άρα να ελαχιστοποιηθεί η πιθανότητα ήττας.

Επομένως, πριν από κάθε κίνηση ο “Pacman” αξιολογεί την κάθε δυνατή του κίνηση 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 μία κίνηση του “Pacman”. Οι μεταβλητές `nodeX, nodeY` περιέχουν τις συντεταγμένες που θα έχει ο “Pacman” έαν ακολουθήσει την κίνηση που υποδεικνύει η μεταβλητή `nodeMove`, ενώ οι μεταβλητές `pacmanX, pacmanY` περιέχουν τις συντεταγμένες του “Pacman” πριν την κίνηση. 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`.

Τώρα ως προς τις υλοποιημένες συναρτήσεις:

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

  Λαμβάνει ως ορίσματα τρεις ακεραίους:

  `x:` κάθετη συντεταγμένη της θέσης του “Pacman” πριν την κίνηση

  `y`: οριζόντια συντεταγμένη της θέσης του “Pacman” πριν την κίνηση

  `move`: προσεχής κίνηση του “Pacman”

  και ένα δισδιάστατο πίνακα τύπου `Room` που θα περιέχει το περιβάλλον παιχνιδιού.

  Αρχικά, υπολογίζει τις νέες συντεταγμένες του “Pacman” μετά την κίνηση `move`, οι οποίες προκύπτουν εάν στις συντεταγμένες x,y εφαρμοστεί η κίνηση `move`. Οι συντεταγμένες της θέσης του “Pacman” πριν την κίνηση `(x,y)` εισχωρούνται στις μεταβλητές `pacmanX, pacmanY`. Στην συνέχεια, ο constructor αποθηκεύει στη μεταβλητή `Maze` της κλάσης το περιβάλλον παιχνιδιού `Maze`. Ακόμα τοποθετεί στις μεταβλητές `currentGhostPos, flagPos, currentFlagStatus` της κλάσης τις εξόδους των `findGhosts(), findFlags(), checkFlags()` αντίστοιχα.

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

- `private 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 possibleGhostPosition()`

  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 μην καταλήγει κοντά σε τοίχους διότι μπορεί να παγιδευτεί από τα φαντάσματα.

  Πρέπει να σημειωθεί ότι σε αυτή τη συνάρτηση προσθέσαμε ένα bias μεγαλύτερης αποφυγής του δεξιά τοίχου από τους υπόλοιπους (-4) και μεγαλύτερης προσέγγισης του κάτω σε σχέση με τον αριστερά (+1), προκειμένου να δώσουμε ένα είδος ασυμμετρίας στην κίνηση του "Pacman", η οποία έχει ως αποτέλεσμα την ελαχιστοποίηση των περιπτώσεων όπου εμφανίζεται μόνιμη ταλάντωση και την αύξηση των περιπτώσεων νίκης με κατάκτηση όλων των σημαίων.Αυτό συμβαίνει, επειδή ο "Pacman", όταν βρίσκεται στο δεξί κομμάτι του ταμπλό, προτιμάει να πλησιάζει στους πάνω και κάτω τοίχους σε σχέση με τον δεξιά αναγκάζοντας τα φαντάσματα να ακολουθήσουν την κίνηση του και στη συνέχεια, όταν ξανακινηθεί προς την αριστερή μεριά του ταμπλό, να τα ωθήσει να μπερδευτούν στις σημαίες του αριστερού τμήματος και έτσι να τα διασπάσει.

  Στο παρακάτω link μπορεί να παρατηρηθεί η καθοριστική επίδραση του εν λόγω bias στην κατάκτηση όλων των σημαιών, που σε διαφορετική περίπτωση θα εξελισόταν σε ταλάντωση και νίκη στα 1000 βήματα. Η κρίσιμη κίνηση είναι στο

σημείο των 53s όταν κινείται το “Pacman” προς τα κάτω και ρίχνει τα φαντάσματα στην σημαία, ξεφεύγοντας έτσι από αυτά:

https://drive.google.com/file/d/1np18KUoJr09TlRNx1giIEBsn9QNqGFjW/view

- `public int getMove()`

  Επιστρέφει την μεταβλητή `nodeMove` 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 στις εξής 2 κατηγορίες:

  1) Κίνηση που καταλήγει σε θέση που ήδη υπάρχει φάντασμα.
  2) Κίνηση που επιστρέφει `true` από την `possibleGhostPosition()` και ταυτόχρονα είναι μία εκ των “πάνω” και “κάτω”. Το πόρισμα αυτό μπορεί να ληφθεί από τη μελέτη της κίνησης των φαντασμάτων. Κάθε φάντασμα, πρώτα προσπαθεί να εκμηδενίσει την οριζόντια απόσταση από τον “Pacman” και έπειτα την κάθετη. Συνεπώς, αν υπάρχει διαφορά και στην οριζόντια και στην κάθετη απόσταση ενώ απέχει από τον “Pacman” απόσταση Μανχάταν 1, τότε σίγουρα το φάντασμα θα εκμηδενίσει την οριζόντια απόσταση. Οπότε το να κινηθεί ο “Pacman” κάθετα, θα τον οδηγήσει με σιγουριά στην ήττα.

  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 50 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:

  Με την επισήμανση ότι για τον δεξιά τοίχο υπάρχει μεγαλύτερο bias να θεωρηθεί κοντινότερος, από ότι ισχύει στην πραγματικότητα

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. Αυτό επιλέχθηκε διότι επιθυμούσαμε ο “Pacman” να είναι αρκετά επιθετικός και επειδή επιπλέον διαπιστώσαμε πειραματικά ότι τα πάει αρκετά καλά στο να αποφεύγει να πέσει πάνω σε φαντάσματα, ανεξάρτητα του πόσο κοντά βρίσκεται σε αυτά. Ένα άλλο κριτήριο, μικρότερης σημαντικότητας, είναι αυτό της αποφυγής των τοίχων, ώστε να μη φτάσει σε σημείο να εγκλωβιστεί από φαντάσματα χωρίς να έχει διαθέσιμες κινήσεις. Τέλος, έχουμε προσθέσει μία αμυδρή επιρροή από την απόσταση από το κοντινότερο φάντασμα. Αυτή είναι στατιστικά σημαντική όταν τα άλλα κριτήρια είναι ανίσχυρα, ή σε περιπτώσεις που δύο θέσεις, βάση των άλλων κριτηρίων, είναι ίδιες.

- Η κλάση `Creature`
  - `public int calculateNextPacmanPosition(...)`
    Η συνάρτηση αυτή εκτελεί τρεις λειτουργίες:

    1. Εντοπίζει τον “Pacman” μέσα στο περιβάλλον `Maze`, το οποίο είναι όρισμα στην στην μέθοδο αυτή.

    2. Υπολογίζει τις δυνατές κινήσεις του “Pacman” από την θέση στην οποία βρίσκεται, δηλαδή ελέγχει σε ποιες από τις τέσσερις (4) κατευθύνσεις (North, South, East, West) δεν υπάρχει τοίχος και μπορεί να κινηθεί ο “Pacman”. Για κάθε μία από τις περιπτώσεις δυνατής κίνησης προσθέτει στο `ArrayList` τύπου `Node89068978` ένα αντικείμενο της κλάσης `Node89068978`, το οποίο ουσιαστικά αναπαριστά την θέση του Pacman μετά την εκάστοτε κίνηση.

    3. Ελέγχει ποιο αντικείμενο του `ArrayList` κατέχει την μεγαλύτερη βαθμολόγηση από την `evaluate()` και επιστρέφει την κίνηση που σχετίζεται με αυτό το αντικείμενο ως την καλύτερη κίνηση, δηλαδή αυτή που θα κάνει ο “Pacman” σε αυτόν τον γύρο. Ουσιαστικά για κάθε δυνατή κίνηση (κάθε αντικείμενο της κλάσης `Node89068978`) η `evaluate()` αξιολογεί την κίνηση αυτή δημιουργώντας μια βαθμολογία ή οποία βασίζεται σε συνεχείς συναρτήσεις και λειτουργεί ως εξής:

Εάν η κίνηση μεγαλώνει την απόσταση από το κοντινότερο φάντασμα, τότε μικραίνει η τιμή που αφαιρείται από το σκορ (καλύτερη βαθμολογία έτσι).

Εάν η κίνηση αυξάνει την απόσταση από την καλύτερη προς κατάκτηση σημαία μειώνεται η τιμή που προστίθεται στην βαθμολογία (αυτό μειώνει την τελική βαθμολογία). Εδώ πρέπει να σημειωθεί ότι η συνάρτηση `bestFlag()` επιστρέφει την ίδια σημαία σε όλα τα αντικείμενα της `ArrayList`. Άρα αυτό το κριτήριο εντοπίζει εάν η συγκεκριμένη πιθανή κίνηση φέρνει τον “Pacman” πιο κοντά ή όχι στον καλύτερο επόμενο στόχο (την καλύτερη σημαία προς κατάκτηση) ο οποίος καθορίστηκε με βάση την τωρινή (πριν την κίνηση) θέση.

Εάν ο “Pacman” κινείται προς τα άκρα μειώνεται η τιμή που προσδίδεται στην βαθμολογία του.

Ανάλογα με το κατά πόσο μεγιστοποιεί ή ελαχιστοποιεί τα παραπάνω κριτήρια κάθε κίνηση λαμβάνει μια συνολική βαθμολογία και στο τέλος επιλέγεται η κίνηση με το μεγαλύτερο σκορ.

Η συνάρτηση αυτή λοιπόν είναι συνάρτηση απόφασης, αφού με βάση τις δυνατές επιλογές κίνησης εντοπίζει ποια είναι η καλύτερη και την ακολουθεί.