ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

3η εργασία

JANUARY 14, 2017

ΠΑΛΑΣΚΟΣ ΑΧΙΛΛΕΑΣ – ΠΑΛΑΣΚΟΣ ΜΑΡΙΟΣ

ΤΑ ΣΤΟΙΧΕΙΑ ΜΑΣ...

ΟΝΟΜΑΤΕΠΩΝΥΜΑ: Παλάσκος Αχιλλέας , Παλάσκος Μάριος

ΑΕΜ: 8493 , 8492

ΤΗΛ: 6946949687 , 6986012613

ΗΛ. ΔΙΕΥΘΥΝΣΕΙΣ: palach2@yahoo.gr mariospala@gmail.com

ΣΤΟΧΟΣ ΤΗΣ ΕΡΓΑΣΙΑΣ

Στόχος της 3ης αυτής εργασίας είναι η δημιουργία ενός παίκτη minmax, δηλαδή σκοπός μας είναι να δημιουργήσουμε έναν παίκτη, ο οποίος:

- Θα λαμβάνει υπόψη του τα ζαχαρωτά που φεύγουν με τα chain moves.
- Η επιλογή της επόμενης κίνησης θα γίνεται με βάση έναν αλγόριθμος minmax (βάθος δεντρου = 2), δηλαδή ο παίκτης μας για κάθε δυνατή κίνησή του θα λαμβάνει υπόψη του όλες τις πιθανές κινήσεις του αντιπάλου.

Table of Contents

Class MinMaxPlayer **3**

int [] getNextMove(Arraylist<int[]> availableMoves,Board board) 3

void createMySubTree(Node84928493 parent, int depth) 4

void createOpponentSubTree(Node84928493 parent, int depth) 5

int chooseMove(Node84928493 root) 6

Class Node84928493 ..... **7**

Contructors - setters - getters 7

int chainMoves(Board board) 8

A) <u>ΚΛΑΣΗ MINMAXPLAYER</u>

int [] getNextMove(Arraylist<int[]> availableMoves,Board board)

- **ΓΕΝΙΚΑ:**
  - Η συνάρτηση αυτή δέχεται ως ορίσματα το αντικείμενο availableMoves τύπου Arraylist που περιέχει το σύνολο των διαθέσιμων κινήσεών μας και το board τη χρονική στιγμή της κίνησής μας.
  - Επιστρέφει έναν πίνακα 1x 3 που περιέχει την καλύτερη κίνηση που έχουμε στη διάθεσή μας με βάση την συνάρτηση αξιολόγησής που υλοποιήθηκε στην 2η εργασία και στη οποία προστέθηκαν τα chain-moves και τον αλγόριθμο minmax.
- **ΔΙΑΔΙΚΑΣΙΑ:**

  Στη συνάρτηση αυτή δημιουργούμε κατ' αρχάς ένα αντίγραφο του board της δεδομένης χρονικής στιγμής, το οποίο ονομάσαμε **nowboard,** μέσω της συνάρτησης *cloneboard* της κλάσης CrushUtilities και με βάση αυτό δημιουργούμε τον κόμβο της ρίζας. Στη συνέχεια δημιουργούμε το δέντρο που περιέχει όλες τις δυνατές κινήσεις μας αλλά και τις αντίστοιχες πιθανές απαντήσεις του αντιπάλου μέσω της κλήσης της συνάστησης createMySubTree() που επιστρέφει τον δείκτη της καλύτερης κίνησής για μας, ο οποίος χρησιμοποιείται ως όρισμα στη συνάρτηση get του availableMoves για να επιστρέψει την καλύτερη κίνησή μας (bestMove). Τέλος, καλείται η *calculateNextMove* με όρισμα τον πίνακα bestMove.

void createMySubTree(Node84928493 parent, int depth)

- **ΓΕΝΙΚΑ:**

  Η συνάρτηση αυτή δέχεται ως ορίσματα τη ρίζα του δέντρου(parent) που είναι και ο πατέρας όλων των παιδιών κόμβων που αντιστοιχούν στις δικές μας κινήσεις και το βάθος των νέων κόμβων(depth) και δημιουργεί το δικό μας υποδένδρο αλλά καλεί και τη συνάρτηση που δημιουργεί τα αντίστοιχα υποδέντρα του αντιπάλου.

- **ΔΙΑΔΙΚΑΣΙΑ:**

  - Αρχικά δημιουργείται το αντικείμενο **myAvailableMoves** τύπου Arraylist και καλείται η συνάρτησή του *getAvailableMoves* με όρισμα το board της ρίζας(parent.getNodeBoard()) για να πάρουμε όλες τις διαθέσιμες κινήσεις μας. Στη συνέχεια για κάθεμία από τις διαθέσιμες κινήσεις μας:
  - Βρίσκουμε το board αμέσως μετά την κίνησή μας (**boardAfterMyMove**), χωρίς να έχουν φύγει ζαχαρωτά, χρησιμοποιώντας την *boardAfterFirstMove* που παίρνει ως ορίσματα το board της ρίζας(parent.getNodeBoard()) και την ι-στή κίνησή μας(myAvailableMoves.get(i)).
  - Δημιουργώ τον εκάστοτε κόμβο (**childNode**) καλώντας τον αντίστοιχο contructor με τα κατάλληλα ορίσματα, δηλαδή με ορίσματα το depth, τον πίνακα κίνησης, boardAfterMyMove, και τον πατέρα κόμβο(που εδώ είναι η ρίζα του δέντρου).
  - Καλείται η setNodeEvaluation του childNode με ορίσματα την κίνηση που αντιστοιχεί στον κόμβο αυτό (childNode.getNodeMove()) και το ταμπλό της ΡΙΖΑΣ (parent.getNodeBoard())

    ΠΡΟΣΟΧΗ!!! : Οταν είχαμε δημιουργήσει τη συνάρτηση αξιολόγησης στη 2η εργασία αυτή έπαιρνε ως ορίσματα την κίνηση και το board **ΠΡΙΝ** την εναλλαγη των πλακιδίων! Γι αυτό εδώ καλούμε την setNodeEvaluation με το board της ρίζας και ΟΧΙ με το board του κόμβου (boardAfterMyMove)!

  - Τέλος, θέτουμε τον κόμβο αυτό ως παιδί της ρίζας με την εντολή: Parent.setChildren(childNode) και καλούμε την createOpponentSubTree με ορίσματα όπως φαίνονται στον κώδικα, διότι για τους κόμβους του αντιπάλου "πατέρας" θα είναι το childNode και το βάθος των νέων κόμβων θα είναι αυτό που έχουν τα childNode +1.

void createOpponentSubTree(Node84928493 parent, int depth)

- **ΓΕΝΙΚΑ:**

  Η συνάρτηση αυτή δέχεται ως ορίσματα το ι-στό παιδί της ρίζας που είναι και ο πατέρας όλων των παιδιών κόμβων που αντιστοιχούν στις κινήσεις του αντιπάλου και το βάθος των νέων κόμβων (depth). Δημιουργεί το υποδένδρο του αντιπάλου και τοποθετεί στα φύλλα τη συνολική αξιολόγηση κάθε δυάδας κινήσεων (δικής μας και του αντιπάλου).

- **ΔΙΑΔΙΚΑΣΙΑ:**
  - Κατ' αρχάς δημιουργούμε το ταμπλό όπως είναι πριν παίξει ο αντίπαλος (**boardBefOpMove**). Αυτό γίνεται καλώντας τη συνάρτηση *boardAfterFullMove* με ορίσματα το ταμπλό πριν τη δική μας κίνηση (parent.getParent.getNodeBoard()) και την εκάστοτε κίνησή μας (parent.getNodeMove()).
  - Αφού δημιουργήσουμε το αντικείμενο **oponAvailableMoves** τύπου Arraylist και βρούμε όλες τις διαθέσιμες κινήσεις του αντιπάλου καλώντας την μέθοδό του *getAvailableMoves* με όρισμα το boardBefOpMove, στη συνέχεια για κάθεμία από τις διαθέσιμες κινήσεις του αντιπάλου:
  - Βρίσκουμε το board αμέσως μετά την κίνηση του αντιπάλου (**boardAfterOpMove**), χωρίς να έχουν φύγει ζαχαρωτά, χρησιμοποιώντας την *boardAfterFirstMove* που παίρνει ως ορίσματα το boardBefOpMove και την j-στή κίνησή του(oponAvailableMoves.get(j)).
  - Δημιουργούμε τον αντίστοιχο κόμβο (**childNode2**) καλώντας τον αντίστοιχο contructor με τα κατάλληλα ορίσματα, δηλαδή με ορίσματα το depth, τον πίνακα κίνησης, **boardAfterOponMove** και τον πατέρα κόμβο(που εδώ είναι το αντίστοιχο childNode).
  - Καλείται η setNodeEvaluation του childNode2 με ορίσματα την κίνηση που αντιστοιχεί στον κόμβο αυτό (childNode2.getNodeMove()) και το boardBefOpMove.

    ΠΡΟΣΟΧΗ!!! : Οταν είχαμε δημιουργήσει τη συνάρτηση αξιολόγησης στη 2η εργασία αυτή έπαιρνε ως ορίσματα την κίνηση και το board **ΠΡΙΝ** την εναλλαγη των πλακιδίων! Γι αυτό εδώ (όπως και στην συνάρτηση createMySubTree) καλούμε την setNodeEvaluation με το boardBefOpMove και ΟΧΙ με το board του κόμβου (boardAfterOponMove)!

  - Αφού κάνουμε την αξιολόγηση, καλούμε την setNodeEvaluation() του childNode2 (χωρίς ορίσματα) που υπολογίζει τη συνολική αξιολόγηση για κάθε ζεύγος δικής μας κίνησης – κίνηση του αντιπάλου και η νέα τιμή τίθεται ως η νέα τιμή της nodeEvaluation του κόμβων του αντιπάλου.
  - Τέλος, θέτουμε τον κόμβο αυτό ως παιδί του αντίστοιχου childNode με την εντολή: parent.setChildren(childNode2).

int chooseMove(Node84928493 root)

- **ΓΕΝΙΚΑ:**
  Η συνάρτηση αυτή δέχεται ως ορίσματα τον κόμβο της ρίζας (root) και με εφαρμογή του αλγορίθμου minmax υπολογίζεται και επιστρέφεται ο δείκτης της κίνησης που θα μας δώσει τη μεγαλύτερη βαθμολογία (**indexBest**).
- **ΔΙΑΔΙΚΑΣΙΑ:**
  - Αρχικά ορίζουμε την **maxEval** που είναι η μέγιστη από όλες τις βαθμολογίες που θα "ανέβουν" στους κόμβους των δικών μας κινήσεων και αρχικοποιείται σε μια πολυ αρνητική τιμή, ώστε να λειτουργήσει ο αλγοριθμος εύρεσης μεγίστου παρακάτω.

  - Στη συνέχεια για κάθε έναν από τους δικούς μας κόμβους, αφού δημιουργήσουμε το αντικείμενο **myNodes** με την εντολή root.getChildren.get(i) υπάρχουν 2 περιπτώσεις :

  - **1η περίπτωση:** Αν το i-στό myNodes δεν έχει παιδιά τότε αυτό σημαίνει ότι για τη συγκεκριμένη δική μας κίνηση ο αντίπαλος ΔΕΝ έχει διαθέσιμες κινήσεις άρα παίρνουμε μόνο εμείς πόντους και γίνεται reset του table. Αυτο ελέγχεται μέσω της εντολής: **if(myNodes.getChildren==0)**. Άρα μας συμφέρει να μην έχει ο αντίπαλος κινήσεις, οπότε σε αυτήν την περίπτωση ως indexBest επιλέγουμε το i , δηλαδή τον αριθμό της κίνησης αυτής, οποίος και επιστρέφεται.

  - **2η περίπτωση:** Αλλιώς, ορίζουμε την **minEval** που είναι η ελάχιστη τιμή των evaluation των φύλλων του i-στού myNodes και αρχικοποιείται στην τιμή του evaluation του πρώτου φύλλου. Στη συνέχεια από τα παιδιά του myNodes, που έχουν οριστεί ως **oponNodes,** βρίσκουμε αυτό που έχει το ελάχιστο evaluation και θέτουμε την ελάχιστη αυτή τιμή ως το evaluation του κόμβου πατέρα, δηλαδή του αντίστοιχου myNodes με την εντολή: **myNodes.setNodeEvaluation(minEval)**. Τέλος, για κάθε myNodes ελέγχουμε αν το evaluation του είναι μεγαλύτερο από την ήδη υπάρχουσα maxEval, οπότε αν ισχύει κάτι τέτοιο αλλάζουμε την maxEval ώστε να έχει τιμή το evaluation του κόμβου αυτού και τον δείκτη indexBest στην τιμή i. Έτσι, μετά τα δύο loop (το ένα αμφωλευμένο στο άλλο) έχουμε υπολογίσει το επιθυμητό indexBest το οποίο και επιστρέφουμε.

B) ΚΛΑΣΗ Node84928493

CONSTRUCTORS

- Έχουμε 2 διαφορετικούς constructors (υπερφόρτωση):
  - ο ένας έχει ως ορίσματα μόνο ένα αντικείμενο τύπου board, αρχικοποιεί τις μεταβλητές που φαίνονται στον κώδικα και καλείται κατα τη δημιουργία της ρίζας.
  - ο άλλος έχεις ως ορίσματα: το depth του κόμβου, την κίνηση που αντιστοιχεί στον κόμβο αυτό, το ταμπλό μετά την κίνηση αυτή και τον πατέρα του κόμβου και αρχικοποιεί τις μεταβλητές όπως φαίνονται στον κώδικα. Καλείται κατά τη δημιουργία των δικών μας κόμβων και των κόμβων του αντιπάλου.

SETTERS

- Η δημιουργία των συναρτήσεων (μία για κάθε μεταβλητή της κλάσης) είναι γνωστή από την πρώτη εργασία, οπότε δεν θα εξηγηθούν περαιτέρω, παρά μόνο η setNodeEvaluation().
- Έχουμε 3 εκδοχές της setNodeEvaluation():
  1. **void setNodeEvaluation(int []move, Board board):** Όταν καλείται με αυτά τα ορίσματα, ανάλογα με το depth του κόμβου, θέτει στη μεταβλητή nodeEvaluation της κλάσης το αποτέλεσμα της moveEvaluation με θετικό ή αρνητικό πρόσημο. Καλείται όταν υπολογίζουμε την αξιολόγηση των δικών μας κινήσεων και του αντιπάλου.
  2. **void setNodeEvaluation(double k):** Αυτή η εκδοχή θέτει το k ως τιμή της nodeEvaluation. Καλείται στον κώδικα όταν θέλουμε να τοποθετήσουμε την ελάχιστη αξιολόγηση των φύλλων ως αξιολόγηση του κόμβου-πατέρα κατά την εφαρμογή του minmax αλγορίθμου.
  3. **void setNodeEvaluation():** Χωρίς ορίσματα καλείται όταν θέλουμε να υπολογίσουμε τη συνολική αξιολόγηση μιας δυάδας κινήσεων (δικής μας και του αντιπάλου), το οποίο είναι ένα από τα τελευταία βήματα του κώδικα πριν ξεκίνησει ο αλγόριθμος minmax.

GETTERS

- Η δημιουργία των συναρτήσεων (μία για κάθε μεταβλητή της κλάσης) είναι γνωστή από την πρώτη εργασία, οπότε δεν θα εξηγηθούν περαιτέρω μιας και δεν παρουσιάζουν καμία απολύτως ιδιαιτερότητα.

int chainMoves (Board board)

- Καλείται από τη MoveEvaluation με board = boardAfterFirstCrush(board, move), δηλαδή θα πάρει σαν board αυτό που προκύπτει μετά την κίνηση μας αλλά πρίν τα chain moves.
- Δημιουργούμε αρχικά δύο vectors με μέγεθος 2, τα οποία δέχονται το αποτέλεσμα των συναρτήσεων: consecutiveHorTiles και consecutiveVerTiles.
- int [] consecutiveVerTiles(int x, int y, int color, Board board): Τη χρησιμοποιήσαμε και στη προηγούμενη εργασία. Δέχεται σαν ορίσματα ένα σημείο (x,y) του board και ένα χρώμα. Για να υπολογίσουμε τα chain moves την καλούμε για ΚΑΘΕ tile(ή κάθε σημείο) του board δίνοντας το χρώμα του tile στο (x,y) και προφανώς το board. Το πρώτο στοιχείο του επιστρέφοντος vector περιέχει ΠΟΣΑ tiles ΙΔΙΟΥ ΧΡΩΜΑΤΟΣ -με το tile στο (x,y)- βρίσκονται ΔΙΑΔΟΧΙΚΑ (και κολλητά του) πάνω και κάτω από αυτό. Αν πλήθος>=3 τότε σημαίνει ότι το συγκεκριμένο tile ΑΝΗΚΕΙ ΣΕ ΚΑΠΟΙΑ ΤΡΙΑΔΑ (τετράδα ή πεντάδα – δεν μας ενδιαφερει) στη στήλη x, οπότε επιστρέφεται στο πρώτο στοιχείο του vector(a1[0]) το πλήθος αυτό. Αν δεν ανήκει σε ν-άδα, τότε επιστρέφεται 0.
- Η ίδια ακριβώς διαδικασία εκτελείται και από την int [] consecutiveHorTiles(int x, int y, int color, Board board) με τη μόνη διαφορά ότι τώρα ελέγχουμε αν το tile ανήκει σε κάποια ν-άδα στη γραμμή που βρίσκεται. Το αποτέλεσμα επιστρέφεται στο a2[0]. Το δεύτερο στοιχείο των vectors το χρησιμοποιήσαμε στη προηγούμενη εργασία και εδώ δε μας ενδιαφέρει.
- Στη συνέχεια ελέχγουμε αν τουλάχιστον ένα από τα δύο αθροίσματα είναι θετικό. Τότε σημαίνει ότι γενικά το tile που βρισκεται στο (x,y) ανήκει τουλάχιστον σε μία ν-αδα (στη γραμμή ή στη στήλη που βρίσκεται). Δε με ενδιάφερει αν θα ανήκει σε μία ή δυο ν-αδες ταυτοχρονα διότι και τότε αυτό το tile «θα φύγει μία μόνο φορά». Άρα το μετράω πάντα μόνο μία φορά. Επομένως αυξάνω το sum++, το οποίο περιέχει ουσιαστικά τα candies που φεύγουν με τα chain moves.
- Συνεπώς μέσα στην επανάληψη ελέχουμε ξεχωριστά για κάθε tile εάν ανήκει σε κάποια/ες ν-αδα/ες οπότε το προσθέτω στο συνολικό αριθμό των candies (sum). Μέσα στην moveEvaluation καλούμε την συνάρτηση chainMoves, το αποτέλεσμα της οποίας προσθέτουμε στη μεταβλητή totalCandies.