ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ
Candy Crush

Εργασία Β:

Heuristic Player

Ράπτης Δημήτριος **Α.Ε.Μ.: 8467** **e-mail: dmraptis@auth.gr**

Παπαγεωργίου Θωμάς **Α.Ε.Μ.: 8577** **e-mail: tompap@live.com**

ΕΙΣΑΓΩΓΗ

Σε αυτή τη δεύτερη καλούμαστε να υλοποιήσουμε μια συνάρτηση αξιολόγησης κάθε κίνησης. Κάθε κίνηση από τις διαθέσιμες θα συνδέεται με μια `double` τιμή η οποία θα υπολογίζοντας λαμβάνοντας υπόψη μας παράγοντες που θα αναλυθούν παρακάτω. Όσο μεγαλύτερη είναι αυτή η τιμή τόσο «καλύτερη» θα είναι η κίνηση για τον παίχτη μας. Έπειτα θα επιλέγεται και θα εφαρμόζεται η κίνηση με την υψηλότερη βαθμολογία.

Μερικά γενικά σχόλια που θα βοηθήσουν στην κατανόηση του κώδικα:

- Μια κίνηση που μετακινεί δύο πλακίδια μπορεί να σχηματίσει ν-άδες μόνο σε συγκεκριμένα σημεία, όχι σε όλη την περιοχή του board. Δηλαδή αν κινηθούμε αριστερά η δεξιά θα πρέπει να ελέγξουμε τις δύο στήλες που βρίσκονται τα πλακίδια και την σειρά τους. Αντίστοιχα αν κινηθούν πάνω κάτω τις δύο σειρές τους και την στήλη τους. Ο υπόλοιπος πίνακας δεν θα έχει όμοια ζαχαρωτά στη σειρά διότι θα έχουν σκάσει στην κίνηση του προηγούμενου παίχτη.
- Έχουν χρησιμοποιηθεί αρκετές `println` για λόγους debugging οι οποίες εμφανίζονται με την μορφή σχολίων στον κώδικα για δύο λόγους. Για την περίπτωση μελλοντική αναδιαμόρφωση του κώδικα ή για την πιθανή ανάγκη ελέγχου ορισμένων μεταβλητών από τον διορθωτή της.
- Έχει γίνει διάσπαση του κώδικα σε αρκετές συναρτήσεις για να είναι πιο ευανάγνωστος και η να περιγράφεται καλύτερα η λειτουργία τους (Ζητείται συγγνώμη εκ των προτέρων για την παρόμοια ονοματολογία..).
- Οι περισσότερες συναρτήσεις δέχονται ως ορίσματα την κίνηση `int[] move` και τον νέο πίνακα `Board newBoard` μετά την εφαρμογή την κίνησης και πριν σκάσουν τα πλακάκια .

ΣΥΝΑΡΤΗΣΕΙΣ ΓΙΑ ΤΗΝ ΕΥΡΕΣΗ ΤΩΝ ΔΙΑΓΡΑΜΕΝΩΝ ΠΛΑΚΙΔΙΩΝ

```
ArrayList<Tile> deletedCandiesAtRow (Board board, int row)
```

Η συνάρτηση αυτή επιστρέφει ένα `ArrayList` με τα πλακίδια που σχηματίζουν ν-αδες σε μια συγκεκριμένη σειρά row , που μας δίνεται σαν όρισμα. Παρατηρείται πως ο μέγιστος αριθμός όμοιων ζαχαρωτών σε μια σειρά μπορεί να είναι 6. Το ιδανικό αυτό σενάρια παρουσιάζεται παρακάτω:

1	2	2	3	2	3	3	4	6	5

1	2	2	2	3	3	3	4	6	5

4-αδα (ή πεντάδα) σχηματίζεται σε σειρά μόνο αν έχουμε κατεύθυνση κίνησης πάνω η κάτω, όπως φαίνεται στο παρακάτω παράδειγμα:

1	6	4	3	2	3	3	1	2	4
2	1	3	6	3	5	3	6	0	5

1	6	4	3	3	3	3	1	2	4
2	1	3	6	2	5	3	6	0	5

Επιστρέφουμε ένα `ArrayList` και όχι απλώς ένας `int` (αρχική υλοποίηση) διότι βοηθάει σε μια άλλη συνάρτησή μας για τον υπολογισμό του ύψους των διαγραμμένων πλακιδίων.

ΥΛΟΠΟΙΗΣΗ:

Η βασική ιδέα είναι:

Παίρνουμε κάθε πλακίδιο σε μια `for` και για το καθένα από αυτά συγκρίνουμε το χρώμα του με το αμέσως επόμενο σε μία `while`. Αν είναι ίδιο τότε το βάζουμε στο **`sameTiles`** `ArrayList`, αλλιώς (ή αν φτάσουμε στο τέλος της σειράς) βγαίνουμε από την while.

Μέτα αν έχουμε ν-αδα τοποθετούμε τα στοιχεία του **sameTiles** στο τελικό **deletedTiles** και δίνουμε το μέγεθος του **sameTiles** σαν τιμή στην μεταβλητη **n_pleOffset.** Η **n_pleOffset** χρησιμεύει ώστε να ξεκινήσουμε την επόμενη for ακριβώς μετά τα ίδια ζαχαρωτά.

Έχουμε χρησιμοποιήσει δύο `ArrayList` γιατί υπάρχει περίπτωση να έχουμε δυο 3-αδες σε μια σειρά.

Επίσης ας εξηγήσουμε και την εντολή:

```
board.giveTileAt(x + compareIndex, row).getColor()

//Η board.giveTileAt επιστρέφει ένα αντικείμενο Tile και από αυτό παίρνω το
//χρώμα του με την συνάρτηση getColor().
```

`ArrayList<Tile> deletedCandiesAtColumn (Board board, int column)`

Είναι όμοια με τη λειτουργία της `deletedCandiesAtRow(Board board, int row)` απλώς για μία στήλη.

`int deletedCandies (int[] move, Board board)`

Η συνάρτηση αυτή επιστρέφει έναν `int` αριθμό που αντιπροσωπεύει τα διαγραμμένα πλακίδια από την κίνηση move.

Δεν λαμβάνονται υπόψη τα πλακίδια που διαγράφονται λόγω των chain moves.

ΥΛΟΠΟΙΗΣΗ:

Τα ίδια ζαχαρωτά όπως είπαμε μπορεί να βρίσκονται σε συγκεκριμένες γραμμές και στήλες, τρεις στο σύνολο για κάθε κίνηση ανάλογα με την κατεύθυνση της.

Για μια `LEFT` ή `RIGHT` κίνηση θα ελέγξω την γραμμή που βρίσκονται τα πλακίδια και τις δυο αντίστοιχες στήλες.

9										
8										
7										
6										
5					3	4				
4										
3										
2										
1										
0										
	0	1	2	3	4	5	6	7	8	9

Δηλαδή θα επιλέξω τις στήλες με `x = 4` και `x = 5` όπως και την σειρά με `y = 5`.

Για μια `UP` ή `DOWN` κίνηση θα ελέγξω την στήλη που βρίσκονται τα πλακίδια και τις δυο αντίστοιχες σειρές.

9										
8										
7										
6										
5										
4								5		
3								6		
2										
1										
0										
	0	1	2	3	4	5	6	7	8	9

Δηλαδή θα επιλέξω τις σειρές με `y = 3` και `y = 4` όπως και την στήλη με `x = 7`.

Για να ελέγξω μια στήλη ή μια σειρά καλώ αντίστοιχα την συνάρτηση

`deletedCandiesAtColumn(board,column)` ή `deletedCandiesAtRow(board,row)`,

που επιστρέφουν ένα `ArrayList` με τα διαγραμμένα ζαχαρωτά. Δηλαδή ο αριθμός των ζαχαρωτών είναι το μέγεθος των `ArrayList`.

ΣΥΝΑΡΤΗΣΕΙΣ ΓΙΑ ΤΗΝ ΕΥΡΕΣΗ ΤΟΥ ΠΡΟΣΑΝΑΤΟΛΙΣΜΟΥ ΤΗΣ ΔΙΑΓΡΑΦΗΣ

```
boolean areDeletedCandiesHorizontal (int[] move, Board board)
```

Η συνάρτηση αυτή επιστρέφει μια boolean τιμή.
True αν τα ίδια πλακάκια είναι οριζόντια και false αν είναι κατακόρυφα.

ΥΛΟΠΟΙΗΣΗ:

Ελέγχει συγκεκριμένες γραμμές ανάλογα με την κατεύθυνση της.

Για μια LEFT ή RIGHT κίνηση θα ελέγξει μόνο την γραμμή που βρίσκονται τα πλακίδια.

Για μια UP ή DOWN κίνηση θα ελέγξει τις δυο σειρές που βρίσκονται τα πλακίδια.

Αν επιστραφεί αριθμός σημαίνει πως έχουν βρεθεί όμοια πλακίδια άρα θα έχουμε μια ν-αδα σίγουρα σε κάποια σειρά.

```
boolean areDeletedCandiesVertical (int[] move, Board board)
```

Είναι όμοια με τη λειτουργία της `areDeletedCandiesHorizontal(Board board, int row)` απλώς για οριζόντιο προσανατολισμό.

```
double deletedCandiesOrientation (int[] move, Board board)
```

Η συνάρτηση αυτή επιστρέφει μια double τιμή.
Οι τιμές που μπορεί να πάρει είναι τρεις:

- 1 : οριζόντια διαγραφή
- 0.5 : κατακόρυφη διαγραφή
- 0 : καμία διαγραφή (δεν υπάρχει τέτοια περίπτωση)

Οι τιμές αυτές λαμβάνονται με την εξής λογική:
Αν μια διαγραφή είναι οριζόντια θα προκαλέσει την εμφάνιση περισσότερων νέων τυχαίων ζαχαρωτών σε σχέση με μια αντίστοιχη

κατακόρυφη. Οπότε αυξάνεται η πιθανότητα ορισμένων chain moves. Για αυτό κα δίνεται μεγαλύτερη τιμή.
Περίπτωση καμίας διαγραφής δεν υπάρχει καθώς η περίπτωση αυτή προϋποθέτει μια κενή `availableMoves`. Τότε όμως δεν θα κληθεί καθόλου η συνάρτηση αξιολόγησης.

ΥΛΟΠΟΙΗΣΗ:

Για την υλοποίηση θα καλέσουμε τις δυο προηγούμενες συναρτήσεις μας `areDeletedCandiesVertical` και `areDeletedCandiesHorizontal` και θα ελέγξουμε τις `boolean` τιμές που θα μας επιστρέψουν.

ΣΥΝΑΡΤΗΣΕΙΣ ΓΙΑ ΤΗΝ ΕΥΡΕΣΗ ΚΟΝΤΙΝΩΝ ΠΛΑΚΙΔΙΩΝ ΙΔΙΟΥ ΧΡΩΜΑΤΟΣ

```
int [] sameColorInProximity (int width, int height,
                             int [] move, Board board)
```

Η συνάρτηση αυτή δέχεται ως ορίσματα δυο `int` μεταβλητές που είναι το πλάτος και το ύψος του χωρίου που θα περικλείσουμε γύρω απτό αρχικό πλακίδιο της κίνησης (`move[0],move[1]`).

Επιστρέφει έναν `int [ ]` πίνακα μεγέθους 7 θέσεων, όσες όλα τα διαφορετικά χρώματα. Σε κάθε κελί αναφέρεται ο αριθμός των πλακιδίων με το ίδιο χρώμα όπως το αντίστοιχο κελί.

πχ. `color[2] = 5` σημαίνει πως έχω 5 ζαχαρωτά χρώματος μπλε στο χώρο που «περιφράξαμε».

Επίσης ως απόσταση (πχ. `width = 3`) του κελιού με `x = 4` ορίζω το εξής:

2	7	1	1	6	4	5	1	5	2

ΥΛΟΠΟΙΗΣΗ:

Καταρχάς θα πρέπει να ελέγξουμε εάν οι τιμές είναι του πλάτους και του μήκους είναι επιτρεπτές και αν όχι να τις προσαρμόσουμε.

Για αυτό το λόγο δημιουργούμε τις μεταβλητές `widthLeft`, `widthRight`, `heightUp`, `heightDown` και τις αρχικοποιούμαι με τις τιμές του αντίστοιχου πλάτους ή ύψους σε κάθε περίπτωση. Αυτό συμβαίνει για να παραμείνουν αυτές οι τιμές σε περίπτωση ΜΗ σφάλματος.

Ο έλεγχος για κάθε μια από αυτές γίνεται ως εξής:

- `widthLeft`:
  το συγκρίνω με την x θέση του αρχικού πλακιδίου `move[0]`. Αν είναι μεγαλύτερο το `width` θέτω ως τιμή του `widthLeft` το `move[0]`.
- `widthRight`:

το συγκρίνω με την μεταβλητή `CrushUtilities.NUMBER_OF_COLUMNS - move[0] – 1` (-1 γιατί θέλω να έχω τιμές από 0 έως 9) που αντιπροσωπεύει την απόσταση του αρχικού πλακιδίου απτό τέλος του πίνακα προς τα δεξιά. Αν είναι μεγαλύτερο το `width` θέτω ως τιμή του `widthRight` το `CrushUtilities.NUMBER_OF_COLUMNS - move[0] – 1.`

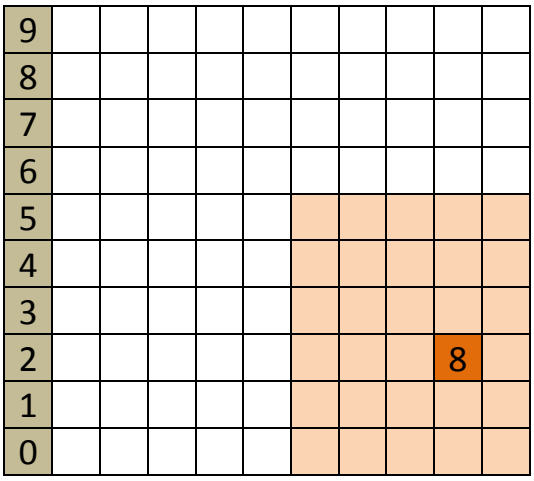
- `heightDown`:
  το συγκρίνω με την y θέση του αρχικού πλακιδίου `move[1].` Αν είναι μεγαλύτερο το `height` θέτω ως τιμή του `heightDown` το `move[1].`
- `heightUp`:
  το συγκρίνω με την μεταβλητή `CrushUtilities.NUMBER_OF_PLAYABLE_ROWS - move[1] – 1` (-1 γιατί θέλω να έχω τιμές από 0 έως 9) που αντιπροσωπεύει την απόσταση του αρχικού πλακιδίου απτό τέλος του πίνακα προς τα πάνω. Αν είναι μεγαλύτερο το `height` θέτω ως τιμή του `heightUp` το `CrushUtilities.NUMBER_OF_PLAYABLE_ROWS - move[1] – 1.`

Έπειτα «σκανάρω» όλο τον πίνακα με δύο `for loops` με μεταβλητές x (αντιστοιχεί στις στήλες) και y (αντιστοιχεί στις γραμμές)

Το x πηγαίνει από `move[0] – widthLeft` έως `move[0] + widthRight` , ενώ

το y πηγαίνει από `move[1] – heightDown` έως `move[1] + heightUp.`

Για παράδειγμα εάν είχα `move[0] = 8` και `move[0] = 2` η περιοχή ελέγχου θα ήταν η εξής:



	0	1	2	3	4	5	6	7	8	9

Για κάθε πλακίδιο θα εκτελεστεί η παρακάτω εντολή:

```
colorNumber [board.giveTileAt (x, y).getColor()] ++;
```

που αυξάνει κατά 1 τον αριθμό στο αντίστοιχο κελί του πίνακά μας (ο αριθμός του κελιού είναι το χρώμα του πλακιδίου, από τον ορισμό της συνάρτησης).

```
double calculateSameColorInProximityCandies (int [] move,
                                             Board board)
```

Η συνάρτηση αυτή επιστρέφει μια double τιμή.

Η τιμή αυτή περιλαμβάνει τον αριθμό των πλακιδίων που βρίσκονται μέσα σε μια περιοχή με `width = height = 3` απτό αρχικό πλακίδιο της κίνησης και επιπλέον έχουν ίδιο χρώμα με τα δυο ζαχαρωτά που αλλάζουν θέση.

Ελέγχω μόνο το χρώμα αυτών των δύο ζαχαρωτών γιατί αυτά είναι περισσότερο πιθανό να σχηματίσουν κάποια n-αδα άμεσα.

ΥΛΟΠΟΙΗΣΗ:

Καλώ την `SameColorInProximityCandies` για να υπολογίσω τα ίδια ζαχαρωτά με το αρχικό πλακίδιο της κίνησης. Ελέγχω την κατεύθυνση της κίνησης και αναλόγως ξανά καλώ την `SameColorInProximityCandies` για να υπολογίσω τα ίδια ζαχαρωτά με το τελικό πλακίδιο.

Επιστρέφω τον συνολικό αριθμό των ίδιων ζαχαρωτών.

ΣΥΝΑΡΤΗΣΕΙΣ ΓΙΑ ΤΗΝ ΕΥΡΕΣΗ ΤΟΥ ΥΨΟΥΣ ΤΗΣ ΔΙΑΓΡΑΦΗΣ

```
int heightOfDeletedCandies (int move[], Board board)
```

Η συνάρτηση αυτή επιστρέφει την απόσταση απτήν σειρά με `x = 0` του κοντινότερου από τα διαγραμμένα ζαχαρωτά.

Αν τα ζαχαρωτά βρίσκονται σε μια σειρά τότε είναι εύκολη η εύρεση του ύψους διότι είναι ο αριθμός x της σειράς.

Αντιθέτως αν τα διαγραμμένα πλακίδια βρίσκονται σε κάποια στήλη, ως ύψος επιστέφουμε το y του πλακιδίου που βρίσκεται πιο κοντά στην σειρά με `x = 0`.

4
6
1
0
0
0
3
4
5
4

Στο παραπάνω παράδειγμα θα επιστραφεί ως ύψος η τιμή 4.

*Θα μπορούσε αυτή η συνάρτηση να σπάσει σε άλλες δύο `heightOfVerticalDelete` και `heightOfHorizontalDelete` .

ΥΛΟΠΟΙΗΣΗ:

Οι μεταβλητές `y1,y2` χρησιμοποιούνται για την περίπτωση κατακόρυφης διαγραφής.

Αρχικά ελέγχουμε τον προσανατολισμό της διαγραφής με τις αντίστοιχες συναρτήσεις που έχουμε υλοποιήσει:

- Οριζόντια διαγραφή:

Η πιο εύκολη από τις δύο περιπτώσεις.

Ελέγχω την κατεύθυνση της κίνησης (για RIGHT-LEFT υπάρχει μόνο μια σειρά, ενώ για UP-DOWN υπάρχουν δυο σειρές που μπορεί να έχουν όμοια ζαχαρωτά) και αναλόγως επιστρέφω την τιμή y της σειράς. Αν έχω δύο σειρές προς έλεγχο ή έχω διαγραφή σε δύο σειρές ταυτόχρονα επιστρέφω αυτήν με την μικρότερη τιμή y. Για αυτό γίνεται πρώτα ο έλεγχος της «χαμηλότερης» σειράς, ώστε αν βρεθεί διαγραφή να μην συνεχίσει.

- Κατακόρυφη διαγραφή:

Η περίπτωση αυτή είναι πιο περίπλοκη από την προηγούμενη.

Το πρόβλημα που παρουσιάζεται είναι πως σε περίπτωση διαγραφής σε δύο στήλες πρέπει να ελέγξω το y του κατώτερου πλακιδίου κάθε διαγραφής, άρα θα πρέπει να υπάρχει αποθήκευση των δύο αυτών τιμών. Γι αυτό και έχουν δημιουργηθεί οι δυο int μεταβλητές y1, y2.

Τα δύο αυτά πλακίδια (στην περίπτωση ταυτόχρονης διαγραφής σε δύο στήλες) θα είναι το πρώτο πλακίδιο της ArrayList sameTilesArray που επιστρέφει η συνάρτηση deletedCandiesAtColumn ,διότι η σάρωση στην συνάρτηση αυτή ξεκινά από y = 0 δηλαδή το πρώτο πλακίδιο που θα εισαχθεί στο ArrayList θα είναι το πρώτο όμοιο.

Ανάλογα με την κατεύθυνση της κίνησης θα υπολογιστούν οι τιμές y1, y2 των δύο στηλών και θα επιστραφεί η μικρότερη (περίπτωση RIGHT-LEFT) ή θα επιστραφεί κατευθείαν η τιμή y του πρώτου πλακιδίου της ArrayList («χαμηλότερου») (περίπτωση UP-DOWN).

*deletedCandiesAtColumn(board, move [0]).get(0).getY() :

Η deletedCandiesAtColumn(board, move [0]) επιστρέφει ένα ArrayList με τα διαγραμμένα πλακίδια, με την get(0) παίρνουμε το πρώτο πλακίδιο (που μας ενδιαφέρει) και με την getY() παίρνουμε την y τιμή του ζαχαρωτού.

*Στην περίπτωση την κίνησης `RIGHT-LEFT` που δεν έχουμε ταυτόχρονη διαγραφή σε δύο στήλες, τότε η μία εκ των `y1, y2` δεν θα αλλάξει τιμή και θα έχει την αρχική της. Η αρχική τιμή που έχουμε δώσει όμως είναι η `CrushUtilities.NUMBER_OF_PLAYABLE_ROWS = 10` που είναι μεγαλύτερη από οποιαδήποτε τιμή μπορεί να παρεί η άλλη μεταβλητή y (0 έως 9). Άρα θα επιστραφεί η μικρότερη, «σωστή» `y`.

Πχ. Εάν η `y1 = CrushUtilities.NUMBER_OF_PLAYABLE_ROWS` (δηλαδή δεν άλλαξε)

και η `y2 = 5`

θα επιστραφεί η `y2` που είναι το ύψος που ζητάμε.

*Ποτέ δεν θα επιστραφεί η τιμή -1 στο τέλος του κώδικα γιατί θα έχουμε οπωσδήποτε μία διαγραφή, όπως αναλύσαμε σε προηγούμενη συνάρτηση.

```
double calculateHeight(int[] move, Board board)
```

Η συνάρτηση αυτή υπολογίζει την συνεισφορά του ύψους στην αξιολόγηση. Καλούμε την `heightOfDeletedCandies` και αναλόγως την τιμή της επιστρέφουμε το ανάλογο «βάρος».

Οι πιθανές τιμές του ύψους είναι:

- > 8: δηλαδή στην κορυφή του πίνακα. Δεν είναι «καλή» διαγραφή γιατί δεν υπάρχει πιθανότητα chain move. Οπότε επιστρέφεται η τιμή -2.
- >6: δηλαδή στην προτελευταία σειρά του πίνακα. Δεν είναι επίσης «καλή» διαγραφή γιατί υπάρχει πολύ μικρή πιθανότητα chain move. Οπότε επιστρέφεται η τιμή -1.
- >3: δηλαδή σχεδόν στην μέση του πίνακα. Δεν έχει ιδιαίτερο ενδιαφέρον. Οπότε επιστρέφεται η τιμή 0.
- >0: δηλαδή στην αρχή του πίνακα, οπότε ανανεώνεται μεγάλο μέρος του πίνακα και το θεωρούμε σαν bonus. Οπότε επιστρέφεται η τιμή 1.
- == 0 δηλαδή στην πρώτη σειρά του πίνακα. Η πιο ενδιαφέρουσα και βοηθητική περίπτωση, διότι ανανεώνεται ο πίνακας από

πάνω έως κάτω (ανάλογα και με τον προσανατολισμό της διαγραφής) και αυξάνεται η πιθανότητα σχηματισμού chain moves . Οπότε επιστρέφεται η τιμή 2.

ΣΥΝΑΡΤΗΣΗ ΑΞΙΟΛΟΓΗΣΗΣ

```
double moveEvaluation (int[] move, Board board)
```

Δημιουργώ μια double τιμή evaluation και έναν πίνακα μετά την εφαρμογή την κίνησης με την εντολή:

```
Board newBoard = CrushUtilities.boardAfterFirstMove(board, move);
```

Στην evaluation προσθέτω τα εξής αποτελέσματα των συναρτήσεων που καλώ με ορίσματα την συγκεκριμένη move και τον νέο πίνακα newBoard:

- `deletedCandiesOrientation (move, newBoard)`:
  επιστρέφει 1 ή 0.5
- `calculateSameColorInProximityCandies (move, newBoard) / 10`:
  επιστρέφει έναν αριθμό έως `(3 + 3)*(3 + 3) = 36` που τον διαιρώ με 10 (< 3.6) για να μην έχει τόσο πολύ μεγάλη επιρροή στην αξιολόγηση.
- `calculateHeight (move,newBoard)`:
  επιστρέφει -1.5, -1, 0, 1 ή 2
- `deletedCandies(move,newBoard) * 2`:
  επιστρέφει τον αριθμό των διαγραμμένων ζαχαρωτών και το διπλασιάζω γιατί είναι το πιο βασικό κριτήριο υπολογισμού της ποιότητας μιας κίνησης.

ΣΥΝΑΡΤΗΣΗ ΕΥΡΕΣΗΣ ΤΟΥ ΔΕΙΚΤΗ ΤΗΣ ΕΠΟΜΕΝΗΣ ΚΙΝΗΣΗΣ ΑΠΟ ΤΙΣ ΔΙΑΘΕΣΙΜΕΣ

```
int findBestMoveIndex (ArrayList<int[]> availableMoves,
                                                Board board)
```

Δημιουργούμε έναν double πίνακα evals με μέγεθος όσο το `availableMoves ArrayList`.

Για κάθε διαθέσιμη κίνηση υπολογίζουμε την αξιολόγηση της :

```
evals[i] = moveEvaluation( availableMoves.get(i), board );
```

και την αποθηκεύουμε στον πίνακα `evals`.

Έπειτα βρίσκουμε την μεγαλύτερη τιμή του `evals` και επιστρέφουμε τον δείκτη αυτού του κελιού