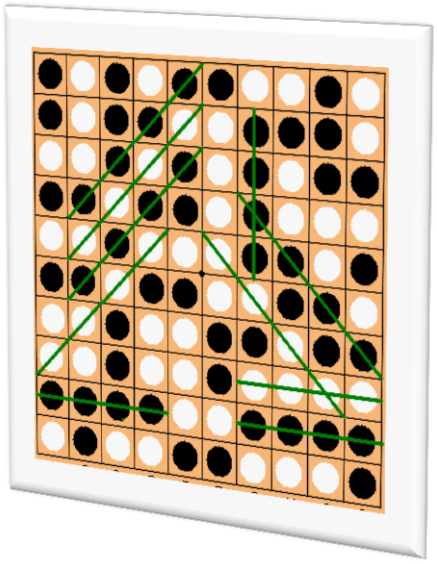
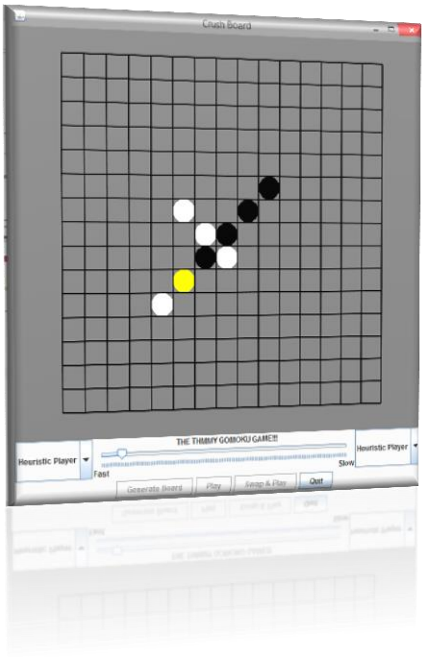
ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
Τμήμα ηλεκτρολόγων μηχανικών & μηχανικών υπολογιστών

ΘΕΜΑ: *«Αναφορά για το δεύτερο μέρος της εργασίας του μαθήματος Δομές Δεδομένων που βασίζεται στο παιχνίδι* **DS – Gomoku** *»*

GOMOKU





ΕΠΙΒΛΕΠΩΝ: ***Μήτκας Περικλής***
Ομάδα εργασίας:

- Triantafyllidis Theocharis/ **Τριανταφυλλίδης Θεοχάρης**
  - **Α.Ε.Μ.:** 7995
  - **E-mail: theocharistr@gmail.com**

Περιγραφή προβλήματος :

Στην δεύτερη παραδοτέα εργασία του μαθήματος καλούμαστε να υλοποιήσουμε μια συνάρτηση αξιολόγησης των διαθέσιμων κινήσεων που έχει ο παίκτης σε κάθε γύρο παιχνιδιού και την αντίστοιχη συνάρτηση επιλογής της καλύτερης κίνησης (δηλαδή σε ποιο πλακίδιο θα τοποθετηθεί το πούλι). Σκοπός μας είναι να δημιουργήσουμε μια όσο το δυνατόν πιο ολοκληρωμένη συνάρτηση αξιολόγησης, που να λαμβάνει υπόψη της τα διαθέσιμα δεδομένα (μέχρι και) εκείνη τη στιγμή και να τα αξιολογεί κατάλληλα, εφαρμόζοντας διάφορα κριτήρια.

Επίλυση προβλήματος :

Για την επίλυση του προβλήματος θα δημιουργήσουμε δυο συναρτήσεις. Αυτές είναι οι :

- `public int[] getNextMove (Board board)`
- `int evaluate (int x, int y, Board board)`

Συνάρτηση *evaluate*

Η συνάρτηση αυτή εκμεταλλεύεται τις εξής συναρτήσεις:

- `public float centrality(int x, int y)`
- `public boolean createsQuintuple(int x, int y ,int color, Board board)`
- `public boolean createsQuatro(int x, int y, Board board,int color)`
- `public boolean createsTriple(int x, int y, Board board,int color)`
- `public boolean createsDouble(int x, int y, Board board,int color)`
- `public int count( int color, int col, int row , Board board,int dirX, int dirY)`

Η συνάρτηση centrality

Η συνάρτηση αυτή δέχεται σαν όρισμα τις συντεταγμένες (x,y) των πλακιδίων και χρησιμοποιείτε για να υπολογίσει την κεντρικότητα του, η οποία παίρνει τιμές στο διάστημα [0,100] και τελικά επιστρέφει την τιμή της κεντρικότητας της θέσης αυτής. Η υλοποίηση της συνάρτησης γίνεται με την προϋπόθεση ότι ο πίνακας (ταμπλό του παιχνιδιού) είναι τετραγωνικός και οι τιμές για τις στήλες και τις γραμμές του πίνακα είναι περιττές. Έχουμε στον νου μας ότι οι συντεταγμένες του κέντρου είναι οι παρακάτω:

- ✓ `x0=((GomokuUtilities.NUMBER_OF_COLUMNS-1)/2)`
- ✓ `y0=((GomokuUtilities.NUMBER_OF_ROWS-1)/2)`

Εφόσον NUMBER_OF_COLUMNS=15 και NUMBER_OF_ROWS=15,που είναι μονοί αριθμοί, για να βρούμε τους μέσους των στηλών και των γραμμών (που ισοδυναμούν με την συντεταγμένη x0 και y0 του κέντρου αντίστοιχα) αφαιρούμε την μονάδα και διαιρούμε με το 2. Οπότε το κέντρο του ταμπλό πρέπει είναι το (7,7). Στην συνέχεια βασισμένοι στον τύπο απόστασης σημείου από σημείο, ο οποίος είναι ο $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$ υπολογίζουμε την απόσταση οποιοδήποτε πλακιδίου από το κεντρικό πλακίδιο (7,7), η οποία στην συνάρτηση μας εκφράζετε με την παρακάτω σχέση:

```
{distance=((((GomokuUtilities.NUMBER_OF_ROWS-1)/2)-y)*(((GomokuUtilities.NUMBER_OF_ROWS-1)/2)-y))
+((((GomokuUtilities.NUMBER_OF_COLUMNS-1)/2)-x)*(((GomokuUtilities.NUMBER_OF_COLUMNS-1)/2)-x));}
```

Εφόσον η sqrt() επιστρέφει double , πρέπει να τον μετατρέψουμε σε float. Για τον λόγο αυτό κάνουμε typecasting:

```
distance=(float)(Math.sqrt(distance));
```

Κατόπιν ,διαιρώ με την μέγιστη δυνατή απόσταση ενός πλακιδίου από το κέντρο. Η διαίρεση γίνεται έτσι ώστε να εξασφαλίσουμε τιμές στο διάστημα [0,1]. Η λογική αυτή υλοποιείται από την εξής σχέση:

```
{distance/=((float)(Math.sqrt((((GomokuUtilities.NUMBER_OF_ROWS)1)/2)*((GomokuUtilities.NUMBER_OF_ROWS-
1)/2) +((GomokuUtilities.NUMBER_OF_COLUMNS-1)/2)*((GomokuUtilities.NUMBER_OF_COLUMNS-1)/2)))); }
```

Επιστρέφουμε το συμπληρωματικό ως προς 1 του αριθμού που πήραμε διότι θέλουμε ο μεγαλύτερος συντελεστής να υποδηλώνει την μικρότερη απόσταση από το κέντρο. Δηλαδή, ισχύει το γεγονός ότι όσο πιο κοντά στο κέντρο είναι το πλακίδιο τόσο μεγαλύτερη είναι η κεντρικότητα και όσο πιο μακριά από το κέντρο τόσο μειώνεται η κεντρικότητα. Τέλος επιστρέφουμε την τιμή αυτή (που αντιπροσωπεύει την κεντρικότητα) αφού πρώτα την πολλαπλασιάσουμε με 100 έτσι ώστε να παίρνω τιμές στο διάστημα [0,100] και να είναι καλύτερα διαχειρίσιμες από την evaluate.

Η συνάρτηση Count

Η συνάρτηση αυτή να μας δίνει την δυνατότητα να απαριθμήσουμε τα πλακίδια που ανήκουν σε έναν παίκτη στην σειρά στις τρείς δυνατές κατευθύνσεις, αφού επιστρέφει τον ακέραιο αριθμό των πλακιδίων στους προσανατολισμούς αυτούς. Δηλαδή:

I. Οριζόντια
II. Κάθετη
III. Διαγώνια (κύρια και δευτερεύουσα)

Την δυνατότητα αυτή της συνάρτησης εκμεταλλεύονται οι συναρτήσεις createsDouble,createsTriple,createsQuatro και η createQouintuple. Η συνάρτηση αυτή δέχεται σαν όρισμα:

1. Τις συντεταγμένες του πλακιδίου (`int col, int row`)
2. Το χρώμα του πλακιδίου (δηλαδή τον παίκτη στον οποίο ανήκει[`( int color`)
3. Την διεύθυνση στην οποία θέλουμε να απαριθμήσουμε (`int dirX, int dirY`)
4. Το ταμπλό του παιχνιδιού (`Board board)`

Ας δούμε πως λειτουργεί τώρα η συνάρτηση. Αρχικά τοποθετούμε έναν μετρητή (`int ct = 1;`) στον οποίο δίνουμε αρχική τιμή ένα. Του αποδίδουμε αυτή την τιμή διότι δεχόμαστε ότι το πλακίδιο για το οποίο γίνεται ο έλεγχος ανήκει στον παίκτη. Μετά εκμεταλλευόμαστε τις μεταβλητές κατεύθυνσης που δέχεται σαν όρισμα για να κάνουμε τον έλεγχο προς μια συγκεκριμένη κατεύθυνση. Η διευθυνσιοδότηση γίνεται ως εξής:

- **Οριζόντιος έλεγχος**

  Θέτουμε dirX=1 και dirY=0 και εξασφαλίζουμε οριζόντιο έλεγχο προς την δεξιά πλευρά του πλακιδίου. Για την αριστερή πλευρά dirX=-1 και dirY=0. Καταφέρνουμε τον έλεγχο αυτό αφού μόνο η συντεταγμένη X αλλάζει.

- **Κάθετος έλεγχος**

  Ορίζουμε dirX=0 και dirY=1 και έχουμε έλεγχο της καθέτου προς τα πάνω και εάν την τροποποιήσουμε σε dirX=0 και dirY=-1 καταμετράμε προς τα κάτω. Τώρα αλλάζει μόνο η Y συνιστώσα για την κάθετη αρίθμηση.

- **Έλεγχος κύριας διαγωνίου**

  Για να εξετάσουμε την πάνω δεξιά κύρια διαγώνιο τοποθετούμε dirX=1 και dirY=1 και για την κάτω αριστερά dirX=-1 και dirY=-1. Έχουμε Δημάδη μεταβολή των X και Y ομόσημα.

- **Έλεγχος δευτερεύουσας διαγωνίου**

  Ωστόσο εάν αλλάζουν ετερόσημα οι συντεταγμένες έχουμε έλεγχο της δευτερεύουσας διαγωνίου. Για την κάτω δεξιά ορίζω dirX=1 και dirY=-1 ενώ για την πάνω αριστερά dirX=-1 και dirY=1

Ακολουθεί ένας συνοπτικός πίνακας για τις τιμές που πρέπει να τοποθετήσουμε στις μεταβλητές κατεύθυνσης dirX και dirY.

Κατεύθυνση	dirX(μεταβολή στην διεύθυνση X)	dirY(μεταβολή στην διεύθυνση Y)
Οριζόντια προς τα δεξιά	**1**	**0**
Οριζόντια προς τα αριστερή	**-1**	**0**
Κάθετη προς τα πάνω	**0**	**1**
Κάθετη προς τα κάτω	**0**	**-1**
Πάνω δεξιά κύρια διαγώνιος	**1**	**1**
Κάτω αριστερά κύρια διαγώνιος	**-1**	**-1**
Κάτω δεξιά δευτερεύουσα διαγώνιος	**1**	**-1**
Πάνω αριστερά δευτερεύουσα διαγώνιος	**-1**	**1**

Κατόπιν αφού έχουμε ορίσει τις μεταβλητές κατεύθυνσης βρίσκουμε τις συντεταγμένες του επόμενου πλακιδίου. Στην συνέχεια με μια while διαπιστώνουμε τα πούλια στην σειρά που ανήκουν στο παίχτη στον προσανατολισμό που έχει οριστεί. Ταυτόχρονα προσέχουμε να μην φύγουμε από τα όρια του πίνακα. Για κάθε πούλι του παίχτη στην ίδια σειρά αυξάνει τον μετρητή και υπολογίζει τις συντεταγμένες του επόμενου πλακιδίου για εξέταση.

```
while ( r >= 0 && r < GomokuUtilities.NUMBER_OF_ROWS && c >= 0 && c <
GomokuUtilities.NUMBER_OF_COLUMNS && board.getTile(c,r).getColor() == color ) {
                    //Το πλακίδιο ανήκει στον παίκτη
                ct++;
                c += dirX;  // Πήγαινε στο επόμενο πλακίδιο σε αυτή την διεύθυνση
                r += dirY;
            }
```

Ωστόσο μέχρι τώρα έγινε έλεγχος στις κύριους προσανατολισμούς. Δηλαδή, εξέταση οριζόντια δεξιά, κάθετα προς τα πάνω, κύρια διαγώνιος πάνω δεξιά και δευτερεύουσα διαγώνιος κάτω δεξιά. Για τους προσανατολισμούς στην αντίθετη κατεύθυνση (οριζόντια αριστερά, κάθετα προς τα κάτω, δευτερεύουσα διαγώνιο πάνω αριστερά και κύρια διαγώνιο κάτω αριστερά) χρησιμοποιούμε τον ίδιο κώδικα αλλάζοντας απλά κάποια πρόσημα μειώνοντας στην συνέχεια το όγκο του κώδικα μας.

```
            C= col - dirX;  // Ψάξε το πλακίδιο στην αντίθετη την διεύθυνση
            r = row - dirY;
            while ( r >= 0 && r < GomokuUtilities.NUMBER_OF_ROWS && c >= 0 && c <
GomokuUtilities.NUMBER_OF_COLUMNS && board.getTile(c,r).getColor()== color ) {
                //Το πλακίδιο ανήκει στον παίκτη
                ct++;
                C -= dirX;   // Πήγαινε στο επόμενο πλκίδιο σε αυτή την διεύθυνση
                r -= dirY;
            }
```

Τελικά η συνάρτηση επιστρέφει την τιμή του μετρητή.

Η συνάρτηση createsQuintuple

Η **public boolean** createsQuintuple(**int** x, **int** y, Board board,**int** color) που δημιουργούμε επιστρέφει true όταν δημιουργούμε πεντάδα και false όταν δεν δημιουργούμε. Υπάρχουν τρείς δυνατότητες δημιουργίας πεντάδας:

1. Δημιουργία οριζόντιας πεντάδας
2. Δημιουργίας κάθετης πεντάδας
3. Δημιουργία διαγώνιας πεντάδας (κύριας και δευτερεύουσας)

Άρα θα πρέπει να δημιουργήσουμε μια συνάρτηση η οποία θα ελέγχει αυτές τις περιπτώσεις. Η συνάρτηση αυτή, όπως προαναφέραμε εκμεταλλεύεται την *count*. Αρχικά δημιουργούμε μια μεταβλητή που θα δέχεται το αποτέλεσμα της *count*.

```
int i=count( color, y, x,board, 1, 0 );
```

Εδώ πρέπει να επισημάνουμε ότι καλούμε την *count* μόνο για τους κύριους προσανατολισμούς αφού μας δίνει η ίδια την δυνατότητα να γίνει έλεγχος και προς τις αντίθετες κατευθύνσεις. Δηλαδή οι τιμές που θα δώσουμε στην count απεικονίζονται στον πίνακα:

Κατεύθυνση	**dirX(μεταβολή στην διεύθυνση X)**	**dirY(μεταβολή στην διεύθυνση Y)**
Οριζόντια προς τα δεξιά	**1**	**0**
Κάθετη προς τα πάνω	**0**	**1**
Πάνω δεξιά κύρια διαγώνιος	**1**	**1**
Κάτω δεξιά δευτερεύουσα διαγώνιος	**1**	**-1**

Τους προσανατολισμούς στην αντίθετη κατεύθυνση (οριζόντια αριστερά, κάθετα προς τα κάτω, δευτερεύουσα διαγώνιο πάνω αριστερά και κύρια διαγώνιο κάτω αριστερά) τους παρέχει η ίδια η συνάρτηση και άρα δεν χρειάζεται να τους καλέσουμε.

Με if κάνουμε έλεγχο για πεντάδες. Για κάθε if που ικανοποιείτε επιστρέφει true διαφορετικά επιστρέφει στο τέλος false.

```
if (i >= 5)
   return true;
if (count( color, y, x,board, 0, 1 ) >= 5)
   return true;
if (count( color, y, x,board, 1, -1 ) >= 5)
   return true;
if (count( color, y,x,board , 1, 1) >= 5)
   return true;

return false;
```

Πρέπει να σημειώσουμε εδώ ότι ελέγχουμε αν γίνεται τουλάχιστον πεντάδα ή και μεγαλύτερη σειριακή ακολουθία πλακιδίων του παίκτη αφού και τότε είναι νικητής.

Οι συναρτήσεις createsdouble,createstriple,createsquatro

Η λογική της υλοποίησης των συναρτήσεων αυτών είναι ολόιδια με αυτή της συνάρτηση createsQuintuple με την διαφορά ότι στις συνθήκες ελέγχου τοποθετούμε 2,3, και 4.

- Π.χ. για την *συνάρτηση* createsQuatro:

```
int i=count( color, y, x,board, 1, 0 );
  if (i == 4)
    return true;
 if (count( color, y, x,board, 0, 1 ) == 4)
    return true;
 if (count( color, y, x,board, 1, -1 ) == 4)
    return true;
 if (count( color, y,x,board , 1, 1) == 4)
    return true;
```

- Π.χ. για την *συνάρτηση* createsTriple:

```
int i=count( color, y, x,board, 1, 0 );
  if (i == 3)
    return true;
 if (count( color, y, x,board, 0, 1 ) == 3)
    return true;
 if (count( color, y, x,board, 1, -1 ) == 3)
    return true;
 if (count( color, y,x,board , 1, 1) == 3)
    return true;
```

- Π.χ. για την *συνάρτηση* createsDouble:

```
int i=count( color, y, x,board, 1, 0 );
if (i == 2)
   return true;
if (count( color, y, x,board, 0, 1 ) == 2)
   return true;
if (count( color, y, x,board, 1, -1 ) == 2)
   return true;
if (count( color, y,x,board , 1, 1) == 2)
   return true;
```

Στις συναρτήσεις αυτές η ανισότητα γίνεται ισότητα αφού η μία περιέχεται στην άλλη για περισσότερα στη σειρά πλακίδια και όλες μαζί είναι εμπεριέχονται στην *createsQuintuple.* Δεν χρειάζεται περαιτέρω ανάλυση αφού έχει γίνει ήδη εκτενείς αναφορά για την λειτουργία τους στην *createsQuintuple*

Evaluate

Εφόσον υλοποιήσουμε τις παραπάνω συναρτήσεις πάμε να υλοποιήσουμε την κύρια συνάρτηση που είναι η *evaluate.* Η int evaluate (int x, int y, Board board) μας επιστρέφει την αξιολόγηση τις θέσεις που στέλνουμε. Έχει σαν ορίσματα τα x,y και το αντικείμενο board της κλάσης Board.Η αξιολόγηση των διαθέσιμων θέσεων γίνεται με τα παρακάτω κριτήρια:

1) Είναι η θέση αυτή δίπλα σε ήδη σχηματισμένη τετράδα; Μήπως σε αυτή την θέση αποκλείω ήδη υπάρχουσα τετράδα του αντιπάλου από το να γίνει πεντάδα;

2) Κεντρικότητα της θέσης. Μια θέση προς το κέντρο του ταμπλό μπορεί να επεκταθεί προς όλες τις κατευθύνσεις.

3) Πόσα δικά σας πλακίδια βρίσκονται σε κάποια απόσταση r από τη θέση x,y που αξιολογείται ανήκουν στον παίχτη (γίνεται έλεγχος για r=2,r=3,r=4)

4) Το ποσοστό των πλακιδίων που ανήκουν στον παίχτη γύρω από εκείνη την περιοχή και σε απόσταση τεσσάρων πλακιδίων.

Η συνάρτηση θα επιστρέφει μια τιμή int μέσα στο διάστημα [0,100], η οποία θα αντιστοιχεί στην αξιολόγηση της κίνησης που εξετάζετε. Όσο καλύτερη για τον παίκτη κρίνεται η τοποθέτηση πλακιδίου στη θέση x,y του ταμπλό, τόσο μεγαλύτερη πρέπει να είναι η τιμή της θέσης αυτής, που επιστρέφει η συνάρτηση αξιολόγησης. Κατόπιν το πρώτο πράγμα που πρέπει να κάνουμε είναι να διαπιστώσουμε αν κάνουμε πεντάδα που προφανώς είναι η καλύτερη δυνατή μας κίνηση και για τον λόγο αυτό της αποδίδουμε την τιμή 100.

```
if(createsQuintuple(x,y,board,color1) ==true){
            return ( 100);}
```

Το χρώμα του παίκτη που υλοποιεί η heuristic καθώς και του αντιπάλου το βρίσκουμε με αυτόν τον τρόπο:

```
int color1=id;
if (color1==1) {color2=2;}
 else{color2=1;}
```

Η επόμενη καλύτερη δυνατή κίνηση που πρέπει να κάνουμε είναι να σταματήσουμε πιθανή ενδεχόμενη πεντάδα του αντίπαλου και για αυτό ξανακαλούμε την `createsQuintuple` αλλά αυτή τη φορά με το χρώμα του αντίπαλου.

```
else if(createsQuintuple(x,y,color2,board) ==true){
                        return ( 99);}
```

Αποδίδουμε στην αξιολόγηση στην περίπτωση αυτή μια χαμηλότερη τιμή(99) έτσι ώστε να μας δίνει πρώτα την δυνατότητα να κάνουμε πεντάδα εμείς και μετά να δίνει προτεραιότητα στο να σταματήσει την πεντάδα του αντιπάλου δεχόμενη μια υψηλή τιμή αλλά όχι και την καλύτερη δυνατή. Η επόμενη περίπτωση είναι να πάρουμε τον μέσο όρο των τιμών των συναρτήσεων που επιστρέφουν οι centrality και colorPercantage(την οποία καλούμε πάντα με ακτίνα 4.

```
else
{double q=0 ;
int c=(int)centrality(x,y);
q=((double)GomokuUtilities.colorPercentage(board, x, y, 4, id));
q=q*100;
```

Επιστρέφουμε τιμές από [0,100] για την colorPercantage. Κατόπιν αποδίδουμε ένα ποσοστό της τάξεως του 0.7 για το ημιάθροισμα των δυο παραπάνω συναρτήσεων.

```
int a =( (int)( 0.7*(q+c))/ (int)2.0)
```

Κάνουμε typecasting γιατί των ημιάθροισμα έχει float και double ενώ εμείς θέλουμε int (ακέραιο) να επιστρέψουμε.

Ταυτόχρονα με αυτές τις συναρτήσεις καλούνται και οι *συναρτήσεις createsdouble, createstriple, createsquatro* και δίνεται προτεραιότητα σε αυτές παρά στον μέσο όρο `int a =( (int) (q+c) / (int)2.0);`.

Στις *συναρτήσεις createsdouble, createstriple, createsquatro* συνυπολογίζουμε και την κεντρικότητα της θέσης και επιστρέφουμε το ημιάθροισμα. Για κάθε μια από τις συναρτήσεις *createsdouble, createstriple, createsquatro* έχουμε αντιστοιχίσει μια τιμή και ένα ποσοστό 0.1 για την κεντρικότητα. Τα ποσοστά και οι τιμές που έχουμε αποδώσει στις *συναρτήσεις createsdouble, createstriple, createsquatro* προέκυψαν μετά από παρατήρηση της λειτουργία του κώδικα, για όσο το δυνατόν καλύτερη λειτουργία του παίκτη που υλοποιεί η heuristic.
Ενδιάμεσα γίνεται χρήση των παρακάτω συναρτήσεων για να γνωρίζουμε πως αξιολογείται η κάθε θέση :

```
System.out.print(x+" ");
System.out.print(y+" ");
System.out.println(a+" ");
```

Τέλος επιστρέφουμε την τιμή με την οποία έχει αξιολογηθεί η συνάρτηση.

```
return (a);//
```

Η συνάρτηση *getNextMove*

Η συνάρτηση `public int[] getNextMove (Board board)` ορίζει ποια θα είναι η επόμενη κίνηση επιλέγοντας την καλύτερη δυνατή θέση με βάση την *evaluate* και μας επιστρέφει τις συντεταγμένες (x,y) του πλακιδίου με την μορφή ενός μονοδιάστατου πίνακα 2 θέσεων ο οποίος περιέχει την καλύτερη δυνατή κίνηση.

```
int[] position = new int[2];
```

Αρχικά δημιουργούμε μια ArrayList που αποθηκεύει πίνακες.

```
ArrayList<int[]> evaluatedPosition = new ArrayList<int[]>();
```

Στην ArrayList θα προσθέσουμε τις τιμές ενός πίνακα ακεραίων τριών θέσεων flagarray (αφού τον δημιουργήσουμε) (`int[] flagarray = new int[3];`) με τιμές [x,y,evaluation] δηλαδή τη θέση (x,y)=(`flagarray[0]=x;, flagarray [1]=y;`) και `flagarray[2]=evaluation=evaluate (x,y,board);` την αξιολόγηση της θέσης x,y. Κάνουμε 2 for για να αξιολογήσουμε κάθε θέση του πίνακα μας που δεν είναι κατειλημμένη. (`if( board.getTile(x,y).getColor()==0)`) . Με τις δυο for ελέγχουμε όλα τα πλακίδια του ταμπλό (15*15=225) και για κάθε ελεύθερο παίρνουμε την αξιολόγηση και την τοποθετούμε στην ArrayList μέσω του ενός μονοδιάστατου πίνακα τριών θέσεων Έπειτα κάνουμε max για να βρούμε τη καλύτερη κίνηση που μπορούμε να εκτελέσουμε, αξιοποιώντας τις ιδιότητες των ArayListς όπως :

```
evaluatedPosition.add(flagarray);
    evaluatedPosition.size()
    evaluatedPosition.get(i)
```

Δημιουργούμε έναν μονοδιάστατο πίνακα τριών θέσεων (`int[] flagarray2 = new int[3];`) ώστε να μπορούμε να βρούμε την θέση της καλύτερης κίνησης. Αντιστοιχίζουμε στον πίνακα αυτό την θέση που παίρνουμε από την `ArrayList` Χρησιμοποιούμε μια for την ανεύρεση τις καλύτερα αξιολογημένης θέσης:

```
for (int i=0;i<thisArrayList.size();i++)
{
 flagarray2=thisArrayList.get(i);
   if (max1<flagarray2[2]){
   max1= flagarray2[2];  k=flagarray2[0];
    l=flagarray2[1];
    }
   }
```

Αφού εκτελεστεί η for έχουμε τον πίνακα position που θα επιστρέψουμε με τιμές (`position[0]=k;, position[1]=l;`) και οι οποίες είναι η τελική θέση όπου θα δεσμεύσει ο παίκτης μας.