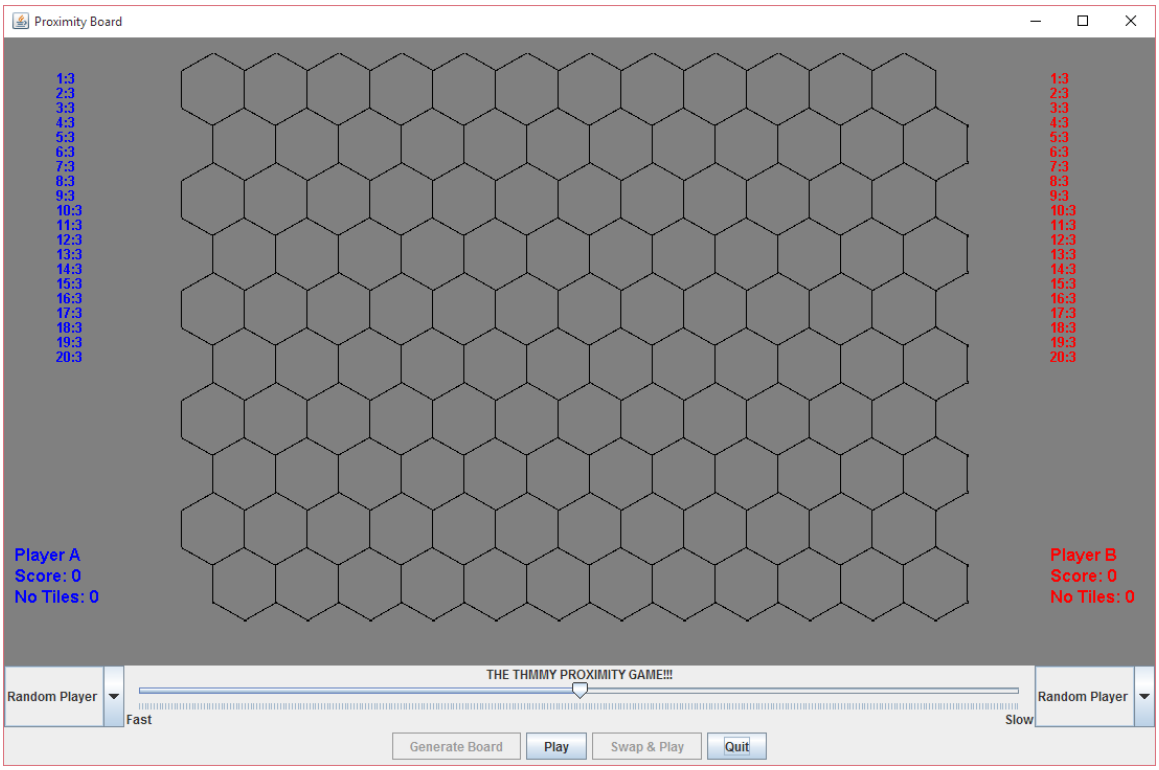
Εργαστήριο Επεξεργασίας Πληροφορίας και Υπολογισμών
Τομέας Ηλεκτρονικής και Υπολογιστών **5ο Εξάμηνο**
ΤΗΜΜΥ Α.Π.Θ **2015-2016**
Τρίτη 1 / 12 / 2015

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

DS – Proximity– Part 2

Heuristic Algorithm (0.75 βαθμοί)

Στο δεύτερο παραδοτέο καλείστε να υλοποιήσετε μια συνάρτηση αξιολόγησης των διαθέσιμων κινήσεων που έχει ο παίκτης σε κάθε γύρο παιχνιδιού και την αντίστοιχη συνάρτηση επιλογής της καλύτερης κίνησης. Σκοπός σας είναι να δημιουργήσετε 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.



Εικόνα 1: Το περιβάλλον του παιχνιδιού DS-Proximity για το μάθημα των Δομών Δεδομένων 2015-2016.

Ο νέος κώδικας της πλατφόρμας περιέχει την υλοποίηση για τον Random Player, η οποία θα είναι ίδια για όλους.

Δεξαμενή Κινήσεων

Σε αντίθεση με το πρώτο παραδοτέο, όπου το σκορ για κάθε πλακίδιο ήταν εντελώς τυχαίο, πλέον κάθε παίκτης έχει μια δεξαμενή από 60 πούλια (3 πούλια για κάθε τιμή από το 1 έως το 20). Πριν από κάθε κίνηση επιλέγεται (από τον υπολογιστή) ένα πούλι με τυχαία τιμή (από όσες είναι διαθέσιμες στην δεξαμενή του παίκτη) και με βάση αυτή την τιμή εκτελείται η επόμενη κίνηση.

Με την χρήση της δεξαμενής διασφαλίζουμε πως παρά την τυχαιότητα της επιλογής της τιμής του πλακιδίου πριν από κάθε κίνηση, τελικά, και οι 2 παίκτες θα έχουν παίξει ακριβώς τα ίδια πλακίδια, απλά με διαφορετική (τυχαία) σειρά.

Χρήσιμες Μεταβλητές και Συναρτήσεις

Για την υλοποίηση των συναρτήσεων που θα σας ζητήσουμε πολύ πιθανόν να χρειαστεί να χρησιμοποιήσετε κάποιες μεταβλητές / συναρτήσεις **που υπάρχουν ήδη υλοποιημένες** στην πλατφόρμα.

Κλάση ***ProximityUtilities***:

- **Στατικές Μεταβλητές της κλάσης *ProximityUtilities***

  Αριθμός Γραμμών και Στηλών:
  NUMBER_OF_ROWS = 10; NUMBER_OF_COLUMNS = 12;

  Κατευθύνσεις:
  Enum Direction{EAST, SE, SW, WEST, NW, NE}
  Χρησιμοποιήστε τις μεταβλητές αυτές όταν θέλετε να βρείτε τον γείτονα ενός πλακιδίου προς μια συγκεκριμένη κατεύθυνση. Δείτε την συνάρτηση **getNeighbor** για περισσότερες λεπτομέρειες.

- **Στατικές Συναρτήσεις της κλάσης *ProximityUtilities***

  int[][] getNeighborsCoordinates(int x, int y, Board board): Επιστρέφει έναν πίνακα 6x2 με τις συντεταγμένες των πλακιδίων που είναι γειτονικά στο πλακίδιο [x,y]. Η σειρά με την οποία αποθηκεύονται τα πλακίδια-γείτονες στον πίνακα είναι αυτή που περιγράφεται στην εκφώνηση της πρώτης εργασίας.

  Tile[] getNeighbors (int x, int y, Board board): Επιστρέφει έναν πίνακα 6x1 με τα πλακίδια που είναι γειτονικά στο πλακίδιο [x,y]. Η σειρά με την οποία αποθηκεύονται τα πλακίδια-γείτονες στον πίνακα είναι αυτή που περιγράφεται στην εκφώνηση της πρώτης εργασίας. Σε αντίθεση με την προηγούμενη συνάρτηση,

getNeighborsCoordinates( ), αυτή η συνάρτηση επιστρέφει έναν πίνακα που περιέχει αντικείμενα τύπου Tile.

Tile getNeighbor (int x, int y, Board board, Direction dir): Επιστρέφει ένα αντικείμενο τύπου Tile το οποίο αναπαριστά τον γείτονα του πλακιδίου στην θέση [x,y] που βρίσκετε στην κατεύθυνση *dir*. Η μεταβλητή *dir* παίρνει συγκεκριμένες τιμές οι οποίες ορίζονται μέσα στο enumeration Direction, όπως περιγράφεται παραπάνω. **Παράδειγμα:** Για να πάρω τον βόρειο-δυτικό γείτονα του πλακιδίου στην θέση [8,7] καλώ την συνάρτηση ως εξής: **Tile tile = ProximityUtilities.getNeighbor(8,7,board,Direction.NW)**

`boolean isInsideBoard(int x, int y)`: Επιστρέφει true αν η θέση [x,y] είναι εντός των ορίων του ταμπλό. Σε διαφορετική περίπτωση επιστρέφει false.

`void printNeighbors(int x, int y)`*:* η συνάρτηση αυτή εκτυπώνει στην κονσόλα το παιχνιδιού τις συντεταγμένες, το χρώμα και το σκορ των πλακιδίων που βρίσκονται γειτονικά του πλακιδίου [x,y]. Αν κάποιος γείτονας δεν υπάρχει εκτυπώνει την τιμή -1.

- **Συναρτήσεις της κλάσης *Tile***

`int getX():` η συνάρτηση αυτή επιστρέφει τη θέση του πλακιδίου στον άξονα x'x.

`int getY():` η συνάρτηση αυτή επιστρέφει τη θέση του πλακιδίου στον άξονα y'y.

`int getColor():` η συνάρτηση αυτή επιστρέφει το χρώμα του συγκεκριμένου πλακιδίου. Συγκεκριμένα έχουμε τα εξής χρώματα:

  - **`0`** → **Grey (**Υποδηλώνει κενή θέση**)**
  - **`1`** → **Blue (**Υποδηλώνει θέση όπου έχει τοποθετήσει πλακίδιο ο παίκτης που παίζει πρώτος**)**
  - **`2`** → **Red (**Υποδηλώνει θέση όπου έχει τοποθετήσει πλακίδιο ο παίκτης που παίζει δεύτερος**)**

`int getScore():` η συνάρτηση αυτή επιστρέφει το σκορ που αντιστοιχεί στο πλακίδιο.

`int playerId():` η συνάρτηση αυτή επιστρέφει τον ιδιοκτήτη του πλακιδίου:

  - **`0`** → **Grey (**Υποδηλώνει πλακίδιο χωρίς ιδιοκτήτη**)**
  - **`1`** → **Blue (**Υποδηλώνει θέση όπου έχει τοποθετήσει πλακίδιο ο παίκτης που παίζει πρώτος**)**
  - **`2`** → **Red (**Υποδηλώνει θέση όπου έχει τοποθετήσει πλακίδιο ο παίκτης που παίζει δεύτερος**)**

- **Συναρτήσεις της κλάσης *Board***

  `Tile getTile(int x, int y):` η συνάρτηση αυτή επιστρέφει το πλακίδιο που βρίσκεται στην θέση [x,y] του ταμπλό.

  `void printBoard():` η συνάρτηση αυτή εκτυπώνει στην κονσόλα το ταμπλό του παιχνιδιού με τη μορφή πίνακα δυάδων-αριθμών της μορφής α/β, όπου α το χρώμα του πλακιδίου όπως φαίνεται στην οθόνη και β το σκορ του πλακιδίου.

  `HashMap<Integer,Integer> getMyPool():` Η συνάρτηση αυτή επιστρέφει ένα αντικείμενο τύπου HashMap που αναπαριστά το σύνολο των διαθέσιμων τιμών που είναι πιθανό να έρθουν για την επόμενη κίνηση. Περισσότερες λεπτομέρειες για το πως να χρησιμοποιήσετε αντικείμενα τύπου HashMap δίνονται παρακάτω.

  `int[] getOpponentsLastMove():` η συνάρτηση αυτή επιστρέφει έναν πίνακα ακεραίων 3 θέσεων της μορφής [x,y,s] όπου x,y οι συντεταγμένες και s το σκορ της προηγούμενης κίνησης του αντιπάλου. Αν δεν υπάρχει προηγούμενη κίνηση, όπως συμβαίνει κατά την πρώτη κίνηση του παιχνιδιού, επιστρέφει την τιμή [-1, -1, -1].

Κλάση ArrayList

Για την υλοποίηση αυτής της εργασίας, **μπορεί** να χρειαστείτε την κλάση ArrayList του πακέτου java.util. Η κλάση αυτή αναπαριστά έναν δυναμικό πίνακα, του οποίου το μέγεθος δεν είναι καθορισμένο από την αρχή, αλλά στον οποίο μπορείτε να προσθαφαιρείτε στοιχεία (αντικείμενα) κατά βούληση. Μπορείτε να δηλώσετε τον τύπο των αντικειμένων που θα περιέχονται. Για παράδειγμα, μια ArrayList με αντικείμενα τύπου String, θα αρχικοποιηθεί ως εξής:

`ArrayList<String> thisArrayList = new ArrayList<String>();`

Με τον ίδιο τρόπο μπορείτε να δηλώσετε και μια ArrayList που αποθηκεύει πίνακες. Σε αυτή την περίπτωση, η δήλωση μιας ArrayList που αποθηκεύει πίνακες ακεραίων θα είναι:

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

Οι μέθοδοι που μπορεί να χρειαστείτε είναι οι εξής:

- `void add(Object o)`: Προσθέτει το αντικείμενο που δίνεται ως όρισμα στην ArrayList.
- `Object get(int pos)`: Επιστρέφει το αντικείμενο που βρίσκεται στη θέση που δώσαμε ως όρισμα. Αν η ArrayList έχει αρχικοποιηθεί για να παίρνει αντικείμενα ενός συγκεκριμένου τύπου, τότε επιστρέφει τον συγκεκριμένο τύπο και όχι ένα απλό Object.
- `int size()`: Επιστρέφει το πλήθος των αντικειμένων που είναι αποθηκευμένα στην ArrayList.
- Μπορείτε να βρείτε και άλλες χρήσιμες μεθόδους στις διαφάνειες του μαθήματος.

Κλάση HashMap

Για την υλοποίηση αυτής της εργασίας, θα χρειαστείτε **οπωσδήποτε** την κλάση HashMap του πακέτου java.util. Η κλάση αυτή αναπαριστά έναν σύνολο από ζευγάρια κλειδιού-τιμής. Είναι δηλαδή μια δομή κατάλληλη για να αποθηκεύει αντιστοιχίες τιμών. Θα πρέπει να δηλώσετε τον τύπο των αντικειμένων που θα περιέχονται τόσο ως κλειδιά όσο και ως τιμές. Για παράδειγμα, ένα HashMap με κλειδιά τύπου ακεραίου (Integer) και τιμές τύπου String θα αρχικοποιηθεί ως εξής:

```
HashMap <Integer,String> testHashMap = new HashMap<Integer,String>();
```

Προσοχή: Όταν ορίζουμε τον τύπο δεδομένων στο HashMap δεν χρησιμοποιούμε τις πρωτόλειες μορφές (int,float,double) αλλά τις αντίστοιχες κλάσεις αντικειμένων (Integer, Double, Float) οι οποίες λειτουργούν ακριβώς με τον ίδιο τρόπο.

Οι μέθοδοι που μπορεί να χρειαστείτε είναι οι εξής:

- `Object put(Object k, Object v):` Προσθέτει το ζευγάρι αντικειμένων που δίνονται ως όρισματα στο HashMap.
- `Object get(Object key):` Επιστρέφει το αντικείμενο που αντιστοιχεί στο κλειδί που δώσαμε ως όρισμα. Αν το HashMap έχει αρχικοποιηθεί για να παίρνει αντικείμενα ενός συγκεκριμένου τύπου, τότε επιστρέφει τον συγκεκριμένο τύπο και όχι ένα απλό Object.
- `int size():` Επιστρέφει το πλήθος των αντικειμένων που είναι αποθηκευμένα στο HashMap.
- Μπορείτε να δείτε και άλλες συναρτήσεις από τις διαφάνειες του μαθήματος.

Συναρτήσεις Προς Υλοποίηση

- **Αξιολόγηση Κενών Θέσεων**

Το πρώτο ζητούμενο είναι να υλοποιήσετε τη συνάρτηση **double getEvaluation(Board board, int randomNumber, Tile tile)** στην κλάση *HeuristicPlayer,* που υπολογίζει πόσο αξιόλογη κίνηση είναι να τοποθετήσετε το πλακίδιο με την τιμή randomNumber στη θέση [x,y] του ταμπλό, τη συγκεκριμένη χρονική στιγμή.

Η αξιολόγηση των διαθέσιμων θέσεων μπορεί να γίνει με διάφορα κριτήρια. Κάποιες **ιδέες** είναι:

1) Πόσο θα αυξηθεί το σκορ μου με την κίνηση που θα κάνω;
2) Θα μειωθεί το σκορ του αντιπάλου;
3) Κεντρικότητα της θέσης. Η θέση που θα τοποθετήσω το πλακίδιο ανήκει σε κάποιο μεγαλύτερο σύμπλεγμα πλακιδίων ή θα είναι απομονωμένο;
4) Πόσους γείτονες θα έχει το πλακίδιο που θα βάλω; Όσο λιγότεροι τόσο πιο εύκολο να κλαπεί από τον αντίπαλο.

Η συνάρτηση που θα φτιάξετε θα πρέπει να επιστρέφει μια τιμή double μέσα σε κάποια λογικά όρια (πχ 0 έως 100), η οποία θα αντιστοιχεί στην αξιολόγηση της κίνησης που εξετάζετε. Όσο καλύτερη

για εσάς κρίνεται η τοποθέτηση πλακιδίου στη θέση [x,y] του ταμπλό, τόσο μεγαλύτερη πρέπει να είναι η τιμή που επιστρέφει η συνάρτηση αξιολόγησης. Αντίστοιχα, όσο χειρότερη – ή καλύτερη για τον αντίπαλο- κρίνεται η τοποθέτηση πλακιδίου από εσάς στην θέση [x,y] τόσο μικρότερη πρέπει να είναι η επιστρεφόμενη τιμή της συνάρτησης.

Για να μην είναι μεροληπτική η αξιολόγηση, ιδανικά θα πρέπει να φροντίσετε να λαμβάνετε υπόψη σας και την θέση που μπορεί να βρίσκεται ο αντίπαλος μετά από εσάς (π.χ. του έχετε αφήσει πολλά πλακίδια με μικρό σκορ που μπορεί εύκολα να κλέψει). **Αυτό το κομμάτι της αξιολόγησης δεν είναι υποχρεωτικό σε αυτή την εργασία.**

Η υλοποίηση αυτής της μεθόδου είναι ελεύθερη, αλλά μπορείτε να βασιστείτε στις γενικές κατευθύνσεις που σας δίνονται παραπάνω. Όμως, ακριβώς επειδή η υλοποίηση είναι ελεύθερη, θα πρέπει να δώσετε ιδιαίτερη βαρύτητα στον σχολιασμό του κώδικα, όσο και στην λεπτομερή ανάλυση του σκεπτικού που ακολουθήσατε στην γραπτή αναφορά που θα παραδώσετε.

Επιπλέον, μπορείτε αν θέλετε να λαμβάνετε υπόψη στην αξιολόγησή σας τα πλακίδια που σας έχουν απομείνει στη δεξαμενή κινήσεων. Μπορείτε, επίσης, να υπολογίσετε και τα πλακίδια που έχουν μείνει στην δεξαμενή του αντιπάλου και να λάβετε και αυτά υπόψη στην αξιολόγησή σας.

ΠΡΟΣΟΧΗ!!!

Θα πρέπει να είστε πολύ προσεκτικοί με τα όρια του ταμπλό (πίνακα). Κατά την διάρκεια της αναζήτησης και της αξιολόγησης υπάρχει μεγάλη πιθανότητα να βγείτε εκτός ορίων, με αποτέλεσμα τον τερματισμό της εκτέλεσης του προγράμματος, αν δεν είστε προσεκτικοί.

- **Επιλογή Διαθέσιμης Κίνησης**

Η συνάρτηση που καλείστε να υλοποιήσετε για την επιλογή της τελικής σας κίνησης είναι η **`int[] getNextMove(Board board, int randomNumber)`** στην κλάση *HeuristicPlayer.* Ο αλγόριθμος υλοποίησης είναι ο εξής:

I. Δημιουργίας μιας δομής (Δυναμικός ή μη πίνακας) που θα αποθηκεύει τις θέσεις [x,y] καθώς και την αξιολόγηση e κάθε μιας από τις πιθανές κινήσεις σας που θα υπολογίσετε.
II. Αξιολόγηση κάθε πιθανής κίνησης με χρήση της συνάρτησης evaluate(Board board, int randomNumber, Tile tile) και αποθήκευση της αξιολόγησης στον πίνακα που έχετε φτιάξει.
III. Επιλογή της καλύτερης κίνησης με βάση την αξιολόγηση που κάνατε.
IV. Επιστροφή ενός μονοδιάστατου πίνακα μεγέθους 3 με τις συντεταγμένες x,y που αντιστοιχούν στην θέση με την καλύτερη αξιολόγηση καθώς και με τον τυχαίο αριθμό που έχετε λάβει ως όρισμα.

Λειτουργία της Πλατφόρμας

- Αρχικά πατάτε το κουμπί που λέει “Generate Board” για να δημιουργηθεί ένα καινούργιο στιγμιότυπο του ταμπλό.
- Στην συνέχεια επιλέγεται τους παίκτες που θα αναμετρηθούν χρησιμοποιώντας τα drop-down boxes στα δύο άκρα της οθόνης. Οι επιλογές είναι 2 για αυτή τη φάση της εργασίας: “Random Player” (που είναι η δική μας υλοποίηση) και “Heuristic Player” που θα είναι ο παίκτης που θα παράγεται από την κλάση που καλείστε να υλοποιήσετε για την δεύτερη εργασία.
- Στη συνέχεια πατάτε το κουμπί “Play” για να ξεκινήσει το παιχνίδι.
- Αν ενδιαφέρεστε να παίξετε αντίπαλοι με τον παίκτη κάποιας άλλη ομάδας, θα πρέπει να κάνετε τα παρακάτω βήματα για να τον φορτώσετε:
    - Δημιουργείτε μια νέα κλάση με το όνομα SecondHeuristicPlayer. (Right click -> New -> Class)
    - Αντιγράφετε τον κώδικα από την κλάση που θέλετε να φορτώσετε στην κλάση που μόλις δημιουργήσατε
    - Αλλάζετε το όνομα της κλάσης και τους constructors σε SecondHeuristicPlayer.
    - Στο drop-down menu ο παίκτης με όνομα Player No2 είναι ο παίκτης μου μόλις εισήγατε.

Οδηγίες

Τα προγράμματα θα πρέπει να υλοποιηθούν σε Java, με πλήρη τεκμηρίωση του κώδικα. Το πρόγραμμά σας πρέπει να περιέχει επικεφαλίδα σε μορφή σχολίων με τα στοιχεία σας (ονοματεπώνυμα, ΑΕΜ, τηλέφωνα και ηλεκτρονικές διευθύνσεις). Επίσης, πριν από κάθε κλάση ή μέθοδο θα υπάρχει επικεφαλίδα σε μορφή σχολίων με σύντομη περιγραφή της λειτουργικότητας του κώδικα. Στην περίπτωση των μεθόδων, πρέπει να περιγράφονται και οι μεταβλητές τους.

Είναι δική σας ευθύνη η απόδειξη καλής λειτουργίας του προγράμματος.

Παραδοτέα για κάθε μέρος της εργασίας

1. Ηλεκτρονική αναφορά που θα περιέχει: εξώφυλλο, περιγραφή του προβλήματος, του αλγορίθμου και των διαδικασιών που υλοποιήσατε και τυχόν ανάλυσή τους. Σε καμία περίπτωση να μην αντιγράφεται ολόκληρος ο κώδικας μέσα στην αναφορά (εννοείται ότι εξαιρούνται τμήματα κώδικα τα οποία έχουν ως στόχο τη διευκρίνιση του αλγορίθμου)
Προσοχή: Ορθογραφικά και συντακτικά λάθη πληρώνονται.

2. Ένα αρχείο σε μορφή .zip με όνομα “AEM1_AEM2_PartB.zip”, το οποίο θα περιέχει **όλο το project σας** στον eclipse καθώς και το αρχείο της γραπτής αναφοράς σε pdf (**αυστηρά**). Το αρχείο .zip θα γίνεται upload στο site του μαθήματος ***στην ενότητα των ομαδικών εργασιών και μόνο***. Τα ονόματα των αρχείων πρέπει να είναι με **λατινικούς χαρακτήρες**.

Προθεσμία υποβολής

Κώδικας και αναφορά Κυριακή 27 Δεκεμβρίου, 23:59 (ηλεκτρονικά)
Δε θα υπάρξει καμία παρέκκλιση από την παραπάνω προθεσμία χωρίς σοβαρό λόγο.