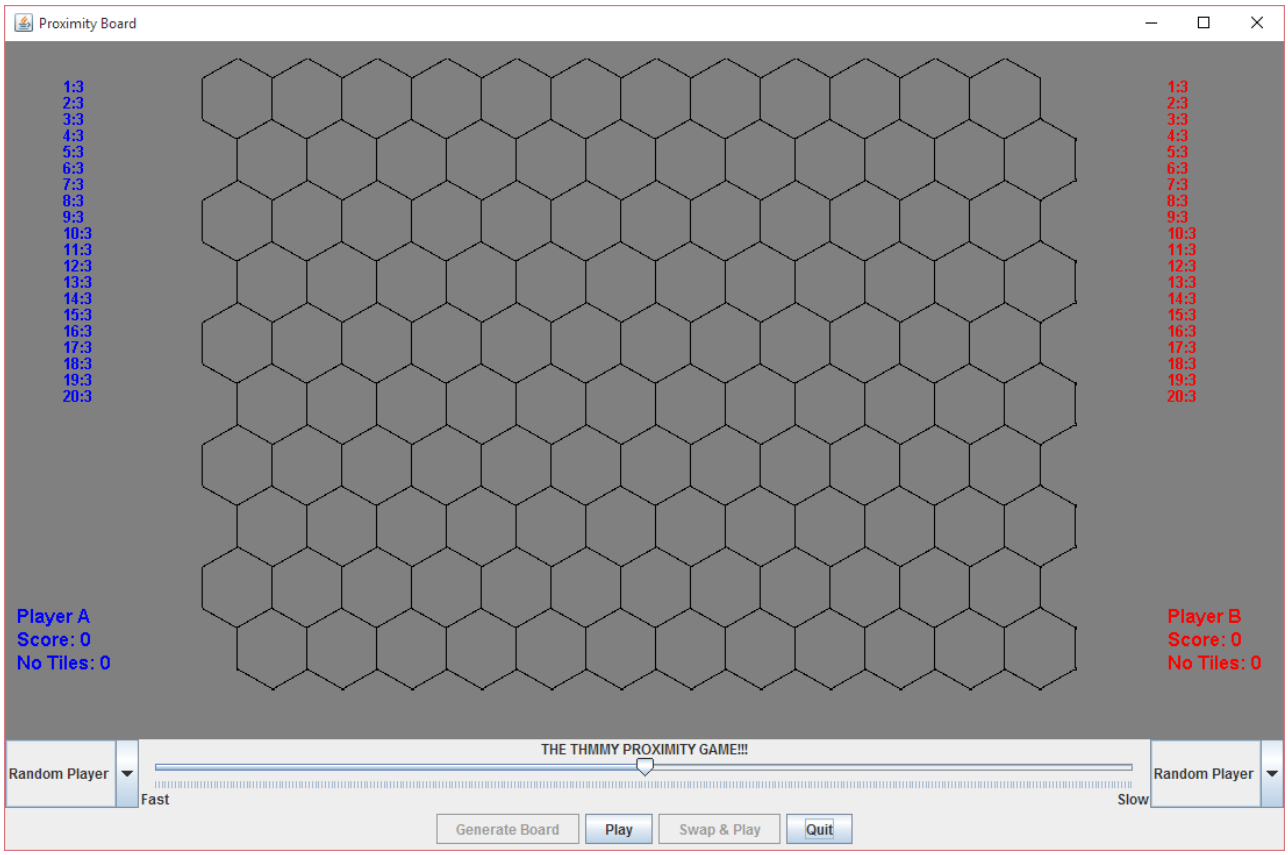
Εργαστήριο Επεξεργασίας Πληροφορίας και Υπολογισμών
Τομέας Ηλεκτρονικής και Υπολογιστών **5ο Εξάμηνο**
ΤΗΜΜΥ Α.Π.Θ **2015-2016**
Πέμπτη 24 / 12 / 2015

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

DS – Proximity – Part 3

MinMax Algorithm (0,5 βαθμοί)

Στο τρίτο παραδοτέο καλείστε να υλοποιήσετε τον αλγόριθμο MinMax για τη βελτιστοποίηση του παίκτη σας. Σκοπός σας είναι να δημιουργήσετε ένα δέντρο βάθους 2 κινήσεων (τουλάχιστον), το όποιο, σε συνδυασμό με τη συνάρτηση αξιολόγησης που είχατε δημιουργήσει στην δεύτερη εργασία, θα αξιολογεί τις διαθέσιμες κινήσεις σε βάθος χρόνου και θα επιλέγει την καλύτερη δυνατή για τον επόμενο γύρο.



Εικόνα 1: Το περιβάλλον του παιχνιδιού DS-Proximity.

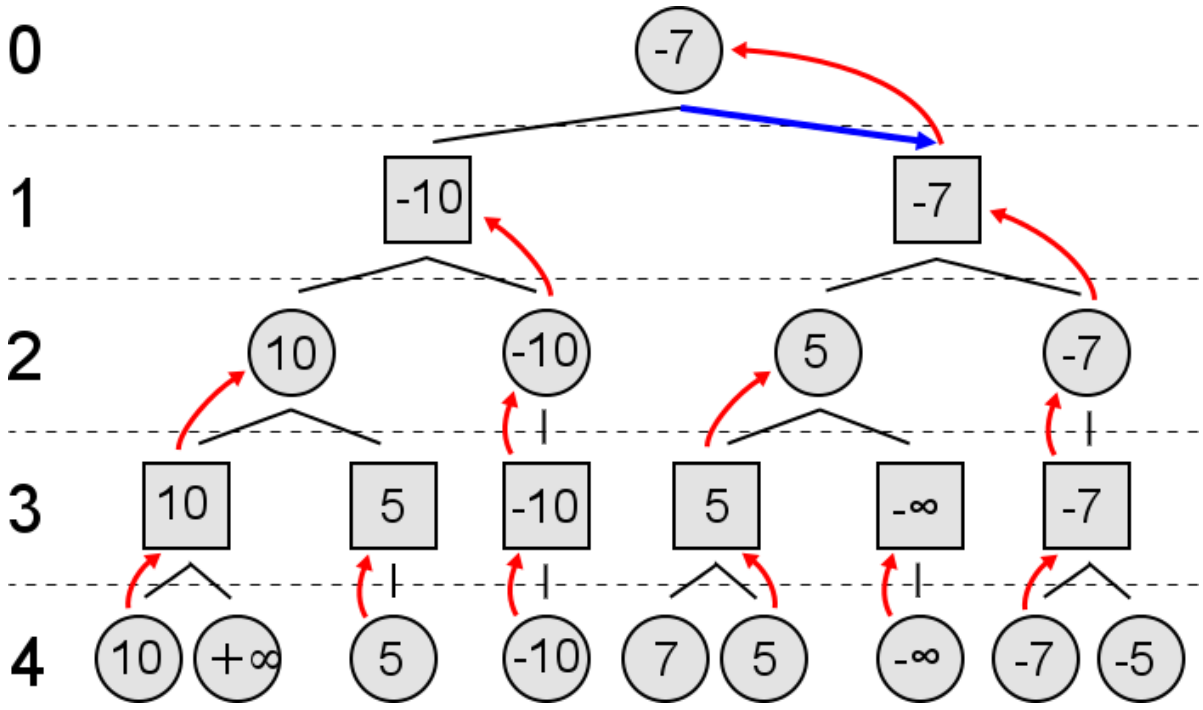
Ο νέος κώδικας της πλατφόρμας περιέχει την υλοποίηση ενός Random Player, καθώς και έναν Heuristic Player με μια τυπική υλοποίηση συνάρτησης αξιολόγησης, η οποία θα είναι ίδια για όλους. Για τη δημιουργία του MinMax Player που καλείστε να σε φέρετε σε πέρας σε αυτό το παραδοτέο, θα χρησιμοποιήσετε τη συνάρτηση αξιολόγησης που υλοποιήσατε στα πλαίσια της δεύτερης εργασίας.

Αλγόριθμος MinMax – AB Pruning

Όπως σε πολλά παιχνίδια, έτσι και στο δικό μας, για τον υπολογισμό της βέλτιστης κίνησης δεν είναι εφικτό να υπολογίσουμε όλες τις διαθέσιμες κινήσεις (μέχρι το τέλος του παιχνιδιού) σε εύλογο χρονικό διάστημα. Εξαιτίας αυτού του γεγονότος, το δένδρο των πιθανών συνδυασμών αναπτύσσεται μέχρι ενός βάθους (π.χ. για βάθος 2 κινήσεων έχω όλες τις δυνατές κινήσεις μου και όλες τις πιθανές κινήσεις-απαντήσεις του αντιπάλου). Σε κάθε φύλλο του δένδρου (στο τέλος της απάντησης του αντιπάλου) εξετάζω τη συνάρτηση αξιολόγησης που υλοποιήθηκε στη δεύτερη εργασία. Σε περίπτωση που ένας κόμβος του δένδρου είναι τελικός (το παιχνίδι τελειώνει), μπορούμε να υπολογίσουμε τον νικητή και να σταματήσουμε εκεί.

Το δένδρο που θα φτιαχτεί θα έχει τη δομή του Σχήματος 2, δίνοντας σε κάθε **φύλλο** του το αποτέλεσμα της συνάρτησης αξιολόγησης. Η δική μας νίκη συμβολίζεται με +∞, ενώ η νίκη του αντιπάλου με -∞. Με τετράγωνα εμφανίζονται οι κινήσεις μας, ενώ με κύκλους οι κινήσεις του αντιπάλου. Θεωρούμε πάντα ότι ο αντίπαλος θα παίξει τη χειρότερη για εμάς κίνηση. Για αυτό επιλέγουμε το **ελάχιστο (Minimum)** της συνάρτησης αξιολόγησης (γραμμές 1 και 3). Εμείς φυσικά θα παίξουμε την κίνηση που **μεγιστοποιεί (Maximum)** τη συνάρτηση αξιολόγησης (γραμμές 0 και 2), λαμβάνοντας ως δεδομένο ότι ο αντίπαλος θα παίξει την καλύτερη κίνηση για αυτόν. Στο Σχήμα, λοιπόν, θα επιλέξουμε τη δεξιά κίνηση γιατί μεγιστοποιεί τη συνάρτηση αξιολόγησης, ασχέτως με το τι θα επιλέξει να παίξει ο αντίπαλος.

Το βάθος του δέντρου μας δείχνει πόσες κινήσεις μπροστά κοιτάμε, και κάθε επίπεδο του δείχνει ακριβώς ποια από τις μελλοντικές κινήσεις εξετάζουμε.



Σχήμα 2: Ο αλγόριθμος MinMax για βάθος 4 κινήσεων.

Κλάση Αποθήκευσης Διαθέσιμων Κινήσεων

Για την υλοποίηση ενός δέντρου διαθέσιμων κινήσεων προτείνεται η δημιουργία μιας κλάσης με το όνομα NodeAEM1AEM2[1]. Η κλάση αυτή θα έχει ως μεταβλητές:

1. **NodeAEM1AEM2 parent**: Ο κόμβος-πατέρας του κόμβου που δημιουργήσατε.
2. **ArrayList<NodeAEM1AEM2> children**: Ο δυναμικός πίνακας που περιλαμβάνει τα παιδιά του κόμβου που δημιουργήσατε.
3. **int nodeDepth**: το βάθος του κόμβου στο δέντρο του MinMax Αλγορίθμου.
4. **int[] nodeMove**: Την κίνηση που αντιπροσωπεύει το Node, σαν πίνακας ακεραίων που περιλαμβάνει το x, το y (και την τιμή του πλακιδίου).
5. **Board nodeBoard**: το ταμπλό του παιχνιδιού για τον συγκεκριμένο κόμβο-κίνηση.
6. **double nodeEvaluation**: Την τιμή της συνάρτησης αξιολόγησης που έχετε δημιουργήσει για αυτή τη κίνηση.

Οι συναρτήσεις της κλάσης που θα πρέπει να υλοποιηθούν είναι:

1. **NodeAEM1AEM2():** Ένας ή περισσότεροι constructors για την κλάση σας, με διαφορετικά ορίσματα.
2. Κατάλληλες συναρτήσεις ***get*** και ***set***.
3. **double evaluate():** η συνάρτηση αυτή θα αξιολογεί την κατάσταση του ταμπλό μετά από την κίνηση (μπορείτε να χρησιμοποιήσετε αυτή που υλοποιήσατε στην δεύτερη εργασία ή μια βελτιωμένη έκδοση της).

Οι συναρτήσεις που είχατε υλοποιήσει στη δεύτερη εργασία ως βοηθητικές για την evaluate() θα αποτελέσουν τη βάση σας και για αυτήν την εργασία, με όσες αλλαγές χρειάζονται για να συμπεριληφθούν και οι νέες μεταβλητές.

Στο package node βρίσκεται μια πρότυπη κλάση με το όνομα Node. Θα πρέπει να τη μετονομάσετε (δεξί κλικ → Refactor → Rename) σύμφωνα με την εκφώνηση και να υλοποιήσετε τις κατάλληλες συναρτήσεις.

[1] Όπου AEM1 και AEM2 ο αριθμός μητρώου των δύο ατόμων κάθε ομάδας.

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. Σε περίπτωση που κάποιος από του γείτονες βρίσκεται εκτός ταμπλό, ο πίνακας στην συγκεκριμένη θέση θα περιέχει την τιμή **null.**

  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.

public static double calculateMeanForPool(HashMap <Integer, Integer> pool): Δέχεται σαν όρισμα ένα HashMap με το σύνολο των διαθέσιμων κινήσεων ενός παίκτη και επιστρέφει τον μέσο όρο τους.

public static double calculateMedianForPool(HashMap <Integer, Integer> pool): Δέχεται σαν όρισμα ένα HashMap με το σύνολο των διαθέσιμων κινήσεων ενός παίκτη και επιστρέφει τον διάμεσό τους.

public static Board boardAfterMove(int playerId, Board board, int x, int y, int s): Η συνάρτηση αυτή προσομοιώνει την κίνηση στις συντεταγμένες x,y με τιμή πλακιδίου s από τον παίκτη με id ίσο με playerId σε ένα αντίγραφο του ταμπλό board. Στο τέλος επιστρέφει ένα αντικείμενο τύπου board που αναπαριστά το πώς θα ήταν το ταμπλό μετά την ολοκλήρωση της κίνησης και αφού γίνουν όλοι οι απαραίτητοι υπολογισμοί στο χρώμα και στην ιδιοκτησία των πλακιδίων που βρίσκονταν ήδη στο ταμπλό.

public static Board boardAfterMove(int playerId, Board board , int[] move): Λειτουργεί όπως και η προηγούμενη συνάρτηση με την διαφορά πως οι συντεταγμένες x,y καθώς και το σκορ s δίνονται σαν όρισμα στην μορφή ενός πίνακα 3 θέσεων της μορφής [x,y,s].

- **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. Αν θεωρείτε πως ο παίκτης σας μπορεί να παίξει σωστά χωρίς να λαμβάνετε υπόψη στην αξιολόγησή σας τις τιμές που θα έχετε διαθέσιμες σε επόμενες κινήσεις, μπορείτε κάλλιστα να μην χρησιμοποιήσετε την συγκεκριμένη συνάρτηση.

HashMap<Integer,Integer> getOpponentsPool(): Λειτουργεί όπως και η getMyPool() με την διαφορά πως επιστρέφει το σύνολο των διαθέσιμων κινήσεων του αντιπάλου.

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].

`int[] getNextTenNumbersToBePlayed():` Η συγκεκριμένη στατική συνάρτηση επιστρέφει έναν πίνακα 10 θέσεων στον οποίο βρίσκονται αποθηκευμένα τα 10 επόμενα νούμερα που θα παιχτούν. Στην θέση 0 βρίσκεται το νούμερο που καλείται να τοποθετήσει στο ταμπλό ο παίκτης που παίζει και καλεί την συνάρτηση. Στην θέση 1 βρίσκεται το νούμερο που θα παίξει ο αντίπαλος στην επόμενη κίνηση και ούτω κάθ'εξής. Χρησιμοποιήστε την για να αξιολογήσετε σωστά τις κινήσεις σε βάθος μεγαλύτερο του 1.

Εισαγωγή Δεύτερου Παίκτη – Κλάση SecondPlayer

Αν θέλετε να εισάγετε τον δικό σας παίκτη Heuristic από το δεύτερο παραδοτέο ή τον παίκτη κάποιου φίλου σας για να παίξετε αντίπαλοι ακολουθήστε τα εξής βήματα:

- Αντικαταστήστε τον κώδικά της κλάσης SecondPlayer με τον κώδικά σας.
- Σε περίπτωση που ο παίκτης που θέλετε να εισάγετε είναι τύπου MinMaxPlayer θα πρέπει να δημιουργήσετε άλλη μια κλάση με το όνομα NodeAEM1AEM2 και να αντιγράψετε σε αυτήν τον κώδικα της αντίστοιχης κλάσης τύπου Node.

Αλγόριθμος Δημιουργίας Δέντρου για βάθος 2 κινήσεων

```
int[] getNextMove (Board board, int randomNumber)
  Use current board to create a new node which corresponds to the root of the tree.
  Call createMySubtree(root, 1)
  // The tree is now finished
  Call the chooseMinMaxMove(Node root)to choose the best available move.
  Return the best move.

void createMySubtree(Node parent, int depth)
  Find the empty tile spots of the board of the parent.
  For each empty tile spot on the board:
        Create a clone of the parent node's board and simulate putting
         a tile on this spot by using boardAfterMove() on the parent node board.
        Create a new node as child of the parent node using new board state.
         Add the node as child of the parent node.
        Complete the tree branches by calling
             createOpponentSubtree(newNode, depth+1)


void createOpponentSubtree(Node parent, int depth)
  Find the empty tile spots of this new state of the board.
  For each empty tile spot for the opponent's turn:
        Create a clone of the parent node's board and simulate putting
         a tile on this spot by using boardAfterMove() on the parent node board.
        Create a new node as child of the parent node using the new board state.
        Evaluate the new leaf node.
        Add the node as child of the parent node.


int chooseMinMaxMove(Node root)
  Implement a minmax algorithm to find the best available move.
  Return the index of the best available move.
```

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_PartC.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 να είναι με **λατινικούς χαρακτήρες**.

Προθεσμία υποβολής
Κώδικας και αναφορά Παρασκευή 15 Ιανουαρίου, 23:59 (ηλεκτρονικά)
Δε θα υπάρξει καμία παρέκκλιση από την παραπάνω προθεσμία.