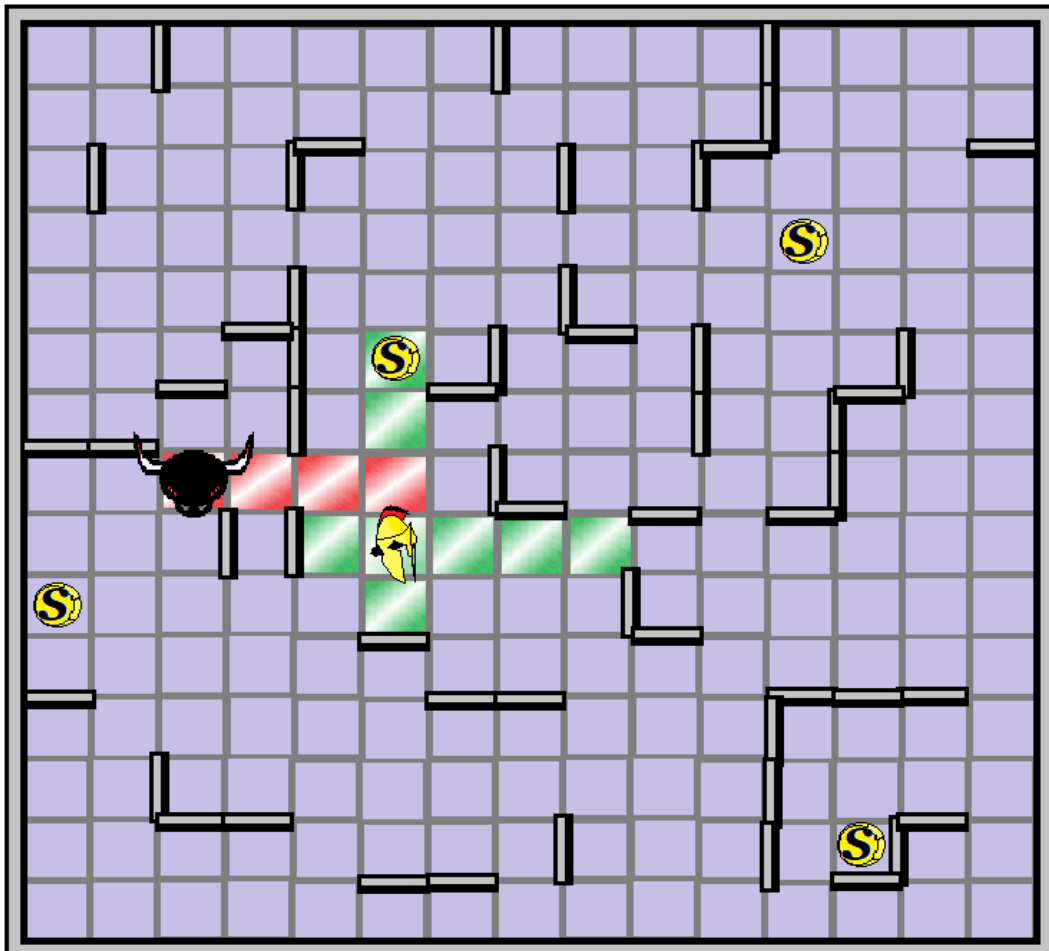


THESEUS VS MINOTAUR

Εργασία Γ



Ταμπλό διάστασης 15x15, με 4 εφόδια

Στοιχεία μελών της ομάδας:

Ονοματεπώνυμο: Τσιανάκας Εμμανουήλ AEM: 9992
κινητό: 6976903077 Email: tsianakas@ece.auth.gr

Ονοματεπώνυμο: Παπούλιας Μιχαήλ AEM: 10204
κινητό: 6949823576 Email: mpapouli@ece.auth.gr

Σύντομη περιγραφή του παιχνιδιού:

Κατά την έναρξη του παιχνιδιού, δημιουργείται ένα ταμπλό διάστασης 15x15 (225 πλακίδια), το οποίο περιέχει 4 εφόδια καθώς και ορισμένο αριθμό τειχών, τυχαία τοποθετημένα στο χώρο. Σε κάθε πλακίδιο μπορούν να τοποθετηθούν το πολύ 2 τείχη, ενώ στο περίγραμμα του ταμπλό υπάρχει υποχρεωτικά τείχος. Ταυτόχρονα, έχουμε και δημιουργία 2 παικτών (του Θησέα και του Μινώταυρου), οι οποίοι μπορούν να κινούνται εναλλάξ πάνω στο ταμπλό. Αρχικά, ο Θησέας βρίσκεται στην κάτω αριστερή γωνία του λαβυρίνθου, ενώ ο Μινώταυρος βρίσκεται στο μέσον του. Οι δύο παίκτες δεν μπορούν να διαπεράσουν τα τείχη, εάν αποφασίσουν να κινηθούν προς μία τέτοια κατεύθυνση. Στόχος του Θησέα είναι να συλλέξει και τα 4 εφόδια προτού τον βρει ο Μινώταυρος. Το παιχνίδι τελειώνει με νικητή το Θησέα, εάν ο ίδιος καταφέρει να συλλέξει τα 4 εφόδια και αποφύγει να συναντηθεί στο ίδιο πλακίδιο με το Μινώταυρο, προτού παρέλθουν 200 γύροι παιχνιδιού. Λήγει με νικητή το Μινώταυρο, εάν καταφέρει να βρεθεί στο ίδιο πλακίδιο με τον Θησέα, πριν τους 200 γύρους. Σε αντίθετη περίπτωση, το παιχνίδι λήγει με ισοπαλία. Σε αυτό το παραδοτέο, η συνάρτηση αξιολόγησης του Θησέα είναι βελτιωμένη, καθώς του επιτρέπει να αξιολογεί τις επιλογές του σε εύρος βάθους δύο κινήσεων (μία του Θησέα και μία του Μινώταυρου). Με τον τρόπο αυτό ο Θησέας καταφέρνει να ξεφεύγει από τον αντίπαλό του, πριν αυτός τον «εντοπίσει».

Κλάση Node:

Η κλάση αυτή αποτελεί τον κόμβο του δένδρου (βάθους δύο επιπέδων) και περιέχει τις εξής μεταβλητές:

Node parent (ένας κόμβος-πατέρας, που περιέχει άλλους κόμβους),

ArrayList<Node> children (οι κόμβοι-παιδιά του δένδρου, αποθηκευμένα σε ένα ArrayList τύπου-Node),

int NodeDepth (το βάθος του δένδρου (σύνολο επιπέδων)),

int[] NodeMove (ένας ακέραιος πίνακας 3 θέσεων, που περιλαμβάνει την τετμημένη, την τεταγμένη και το ζάρι του MinMax παίκτη),

Board nodeBoard (το ταμπλό του παιχνιδιού για μία συγκεκριμένη κομβο-κίνηση),

double nodeEvaluation (η αξιολόγηση για μία συγκεκριμένη κομβο-κίνηση).

Η Node περιέχει 2 κατασκευαστές (constructors). Ο πρώτος αρχικοποιεί τις int και double μεταβλητές στο “μηδέν” και δημιουργεί ένα ArrayList<Node> children, καθώς και έναν ακέραιο nodeMove πίνακα 3 θέσεων. Ο δεύτερος κατασκευαστής λαμβάνει ως όρισμα και δημιουργεί ένα αντικείμενο a τύπου Node. Τέλος, η κλάση αυτή περιέχει τους αντίστοιχους getters και setters όλων των μεταβλητών της.

Κλάση MinMaxPlayer:

Η κλάση αυτή αντιπροσωπεύει τον έξυπνο παίκτη (στο συγκεκριμένο παραδοτέο, ο Θησέας έχει την ιδιότητα αυτή) και έχει τις εξής μεταβλητές:

ArrayList<Integer[]> path (ένα Arraylist που δέχεται ως ορίσματα πίνακες ακεραίων και περιέχει πληροφορίες για την κάθε κίνηση του παίκτη κατά τη διάρκεια του παιχνιδιού.

int count1 (οι φορές του παίκτη που κινήθηκε πάνω)

int count3 (οι φορές του παίκτη που κινήθηκε δεξιά)

int count5 (οι φορές του παίκτη που κινήθηκε κάτω)

int count7 (οι φορές του παίκτη που κινήθηκε αριστερά)

Στην κλάση αυτή προσθέσαμε και μια ακέραια μεταβλητή την int thturns, η οποία μετράει τις φορές που παίζει ο Θησέας.

Η MinMaxPlayer περιέχει 3 κατασκευαστές (constructors). Ο πρώτος αρχικοποιεί τις ακέραιες μεταβλητές στο “μηδέν” και δημιουργεί ένα κενό ArrayList path. Ο δεύτερος κατασκευαστής λαμβάνει ως όρισμα όλες τις μεταβλητές της κλάσης αυτής και τις

προσπελάζει με τη χρήση του τελεστή “this”, ενώ ο τρίτος λαμβάνει ως όρισμα και δημιουργεί ένα αντικείμενο a τύπου MinMaxPlayer. Τέλος, η κλάση αυτή περιέχει τους αντίστοιχους getters και setters όλων των μεταβλητών της.

-Μέθοδος OpponentDist()

Η μέθοδος αυτή συμμετέχει στην επιλογή της καλύτερης κίνησης του παίκτη, ελέγχοντας εάν ο αντίπαλος είναι κοντά, πραγματοποιώντας κίνηση του παίκτη προς μία συγκεκριμένη κατεύθυνση. Οι δυνατές κατευθύνσεις είναι τέσσερις, οπότε η OpponentDist χωρίζεται σε τέσσερις υπό-μεθόδους (εδώ θα αναλύσουμε μόνο την πρώτη, καθώς και οι υπόλοιπες ακολουθούν την ίδια λογική):

OpponentDist1()

(έλεγχος θέσης αντιπάλου προς την κατεύθυνση 1(πάνω)):

Δέχεται δύο ορίσματα τα th (η θέση του Θησέα) και min (η θέση του Μινώταυρου) και περιέχει ένα όρισμα:

int const_th (που διατηρεί την τωρινή θέση του Θησέα έπειτα από τις δοκιμές).

Αρχικά, εάν υπάρχει τείχος στο πάνω μέρος του πλακιδίου που βρίσκεται ο Θησέας, η συνάρτηση επιστρέφει τον αριθμό -10 (ώστε να μην κινείται ποτέ πάνω σε τείχη). Εάν αυτό δεν ισχύει, εισέρχεται σε μια μεγάλη “if” που πραγματοποιεί διαδοχικούς ελέγχους και αρχικά ελέγχει εάν ο Μινώταυρος βρίσκεται στο πάνω πλακίδιο από αυτό που βρίσκεται τώρα ο Θησέας. Αν ισχύει, επιστρέφει την τιμή -1(ώστε να αποφύγει τον αντίπαλο), αλλιώς πραγματοποιεί ελέγχους για το δεύτερο και τρίτο πλακίδιο πάνω από αυτό που βρίσκεται ο Θησέας, και εάν ισχύει κάποιος έλεγχος και δεν υπάρχουν ενδιάμεσα τείχη επιστρέφει -0.5 και -0,3 αντίστοιχα. Όμοια και για τις OpponentDist3, OpponentDist5 και OpponentDist7 .

-Μέθοδος NearSupplies()

Η μέθοδος αυτή συμμετέχει στην επιλογή της καλύτερης κίνησης του παίκτη, ελέγχοντας εάν υπάρχει κοντινό εφόδιο, πραγματοποιώντας κίνηση του παίκτη προς μία συγκεκριμένη κατεύθυνση. Οι δυνατές κατευθύνσεις είναι τέσσερις, οπότε η NearSupplies χωρίζεται σε τέσσερις υπό-μεθόδους (εδώ θα αναλύσουμε μόνο την πρώτη, καθώς και οι υπόλοιπες ακολουθούν την ίδια λογική):

NearSupplies1()

(έλεγχος ύπαρξης εφοδίου προς την κατεύθυνση 1(πάνω)):

Δέχεται ένα όρισμα, το th (η θέση του Θησέα) . Αρχικά, εάν υπάρχει τείχος στο πάνω μέρος του πλακιδίου που βρίσκεται ο Θησέας, η συνάρτηση επιστρέφει τον αριθμό -10 (ώστε να μην κινείται ποτέ πάνω σε τείχη). Εάν αυτό δεν ισχύει, εισέρχεται σε μια μεγάλη "if" που πραγματοποιεί διαδοχικούς ελέγχους και αρχικά ελέγχει εάν ένα από τα εφόδια βρίσκεται στο πάνω πλακίδιο από αυτό που βρίσκεται τώρα ο Θησέας. Αν ισχύει, επιστρέφει την τιμή 1 (ώστε να συλλέξει το εφόδιο) , αλλιώς πραγματοποιεί ελέγχους για το δεύτερο και τρίτο πλακίδιο πάνω από αυτό που βρίσκεται ο Θησέας και εάν ισχύει κάποιος έλεγχος και δεν υπάρχουν ενδιάμεσα τείχη επιστρέφει 0.5 και 0,3 αντίστοιχα. Όμοια και για τις NearSupplies3, NearSupplies5 και NearSupplies7 .

-Μέθοδος evaluate()

Η συνάρτηση αυτή αξιολογεί τις δυνατές κινήσεις του MinMaxPlayer μέσω της συνάρτησης:

$f(\text{NearSupplies}, \text{NearOpponent}) = \text{NearSupplies} * 0,46 + \text{OpponentDist} * 0,54$

Και δέχεται τρία ορίσματα:

int currentpos (η θέση του Θησέα)

int oppentpos (η θέση του Μινώταυρου) και

int dice (η ζαριά)

Με βάση την ζαριά του παίκτη καλούνται οι συναρτήσεις OpponentDist και NearSupplies της αντίστοιχης κατεύθυνσης και επιστρέφεται η συνολική αξιολόγηση κάθε δυνατής κίνησης του Θησέα, σε μια double μεταβλητή.

-Μέθοδος evaluate_minotaur()

Η συνάρτηση αυτή αξιολογεί τις δυνατές κινήσεις του Μινώταυρου μέσω της συνάρτησης:

f(NearOpponent) = - OpponentDist

Και δέχεται δύο ορίσματα:

int currentpos (η θέση του Θησέα)

int oppentpos (η θέση του Μινώταυρου)

Με βάση τις διαθέσιμες κινήσεις του Μινώταυρου, καλείται η συνάρτηση OpponentDist της αντίστοιχης κατεύθυνσης και επιστρέφεται η μέγιστη αξιολόγηση των δυνατών κινήσεων του Μινώταυρου, σε μια double μεταβλητή.

-Μέθοδος chooseMinMaxMove()

Η συνάρτηση αυτή δέχεται τρία ορίσματα:

Node root (ο κόμβος-πατέρας του δέντρου)

int th (η θέση του Θησέα)

int min (η θέση του Μινώταυρου)

και υλοποιεί τον MinMax-αλγόριθμο σε ένα δέντρο βάθους 2 επιπέδων (το πρώτο επίπεδο αφορά την αξιολόγηση του Θησέα και το δεύτερο την αξιολόγηση του Μινώταυρου). Ο MinMax-αλγόριθμος βάθους δύο κινήσεων θα περιγράφεται από την εξής σχέση:

Root-evaluation = max[min[evaluate()-evaluate_minotaur()]]

που υποδηλώνει ότι ο MinMaxPlayer θεωρεί ότι ο αντίπαλός του θα κάνει την χειρότερη (για τον MinMaxPlayer) κίνηση, επομένως επιλέγονται από το δεύτερο επίπεδο του δέντρου οι μικρότερες αλγεβρικές τιμές της διαφοράς [evaluate()-evaluate_minotaur()] και αποθηκεύονται στο αντίστοιχο parent τους (δηλαδή στο πρώτο επίπεδο). Τώρα, ο MinMaxPlayer θα επιλέξει τη βέλτιστη κίνηση για τον ίδιο, από το επίπεδο 1, και αυτή η τιμή θα αποθηκευτεί στη ρίζα του δέντρου.

Τέλος, η συνάρτηση επιστρέφει τη ζαριά της καλύτερης κίνησης του Θησέα (με τη βοήθεια ενός δείκτη index_1), η οποία είναι αποθηκευμένη στη τρίτη θέση του ακέραιου NodeMove[] πίνακα.

-Μέθοδος getNextMove()

Η συνάρτηση αυτή δέχεται ως ορίσματα 3 μεταβλητές:

int currentPos (θέση του Θησέα)

int opponentPos (θέση του Μινώταυρου) και

Player b (ο παίκτης Μινώταυρος)

Επιστρέφει τη νέα θέση του MinMaxPlayer, καλώντας τις συναρτήσεις createMySubtree και chooseMinMaxMove. Δημιουργεί έναν ακέραιο μονοδιάστατο πίνακα 4 στοιχείων, καθένα από τα οποία αντιπροσωπεύει τα εξής:

1ο στοιχείο: (path_array[0]) η ζαριά που αντιστοιχεί στην καλύτερη συνολική αξιολόγηση (το στοιχείο που επιστρέφει η chooseMinMaxMove, δηλαδή το ζάρι).

2ο στοιχείο: (path_array[1]) η συλλογή ή μη ενός από τα 4 εφόδια από τον Θησέα. Σε περίπτωση που ο Θησέας με την συγκεκριμένη κίνησή του συνέλλεξε ένα οποιοδήποτε εφόδιο, το 2^ο στοιχείο λαμβάνει την τιμή 1, διαφορετικά λαμβάνει την τιμή 0.

3ο στοιχείο: (path_array[2]) η απόσταση του Θησέα από το κοντινότερο εφόδιο, μετά την κίνησή του. Οι δυνατές τιμές του 3^{ου}

στοιχείου είναι: 3, 2 ή 1 (εφόσον το εφόδιο βρίσκεται στο οπτικό πεδίο του Θησέα), διαφορετικά το 3^ο στοιχείο θα έχει τιμή -1 (εφόδιο εκτός οπτικού πεδίου του Θησέα).

4ο στοιχείο: (path_array[3]) η απόσταση του Θησέα από τον αντίπαλο, μετά την κίνησή του. Οι δυνατές τιμές του 4^{ου} στοιχείου είναι: 3, 2 ή 1 (εφόσον ο αντίπαλος βρίσκεται στο οπτικό πεδίο του Θησέα), διαφορετικά το 4^ο στοιχείο θα έχει τιμή -1 (αντίπαλος εκτός οπτικού πεδίου του Θησέα).

Στη συνέχεια, γίνεται η κίνηση του Θησέα-MinMaxPlayer. Όταν ο Θησέας κινηθεί, θα συμβούν αλλαγές στις συντεταγμένες του πλακιδίου που βρίσκεται καθώς και στη διεύθυνση του νέου πλακιδίου. Πιθανές αλλαγές θα συμβούν και στο ταμπλό του παιχνιδιού (π.χ ο Θησέας συνέλεξε ένα εφόδιο). Στη περίπτωση αυτή, οι συντεταγμένες του συγκεκριμένου εφοδίου μηδενίζονται, το score του παίκτη αυξάνεται κατά ένα και το δεύτερο στοιχείο του path_array (path_array[1]) λαμβάνει την τιμή 1 (διαφορετικά, η τιμή του παραμένει 0).

Μετά τη νέα κίνηση του MinMaxPlayer, πρέπει να γίνει έλεγχος εάν ένα εφόδιο βρίσκεται ή όχι στο οπτικό πεδίο του. Επομένως, καλούνται διαδοχικές “if”, οι οποίες ελέγχουν την τιμή όλων των NearSupplies συναρτήσεων – εάν αυτές έχουν την τιμή 0.3, τότε ο Θησέας απέχει απόσταση 3 από το κοντινότερο εφόδιο, άρα path_array[2] = 3. Εάν έχουν την τιμή 0.5, τότε απέχει απόσταση 2 από το κοντινότερο εφόδιο, άρα path_array[2] = 2, ενώ, εάν έχουν την τιμή 1, τότε απέχει απόσταση 1 από το κοντινότερο εφόδιο, άρα path_array[2] = 1. Διαφορετικά, ο Θησέας δεν εντοπίζει κανένα εφόδιο μετά τη νέα του κίνηση (path_array[2] = -1):

```
path_array[2] = -1;
if((NearSupplies1(currentPos) == 0.3) || (NearSupplies3(currentPos) == 0.3) ||
   (NearSupplies5(currentPos) == 0.3) || (NearSupplies7(currentPos) == 0.3) )
    path_array[2] = 3;
if((NearSupplies1(currentPos) == 0.5) || (NearSupplies3(currentPos) == 0.5) ||
   (NearSupplies5(currentPos) == 0.5) || (NearSupplies7(currentPos) == 0.5) )
    path_array[2] = 2;
if((NearSupplies1(currentPos) == 1) || (NearSupplies3(currentPos) == 1) ||
   (NearSupplies5(currentPos) == 1) || (NearSupplies7(currentPos) == 1) )
    path_array[2] = 1;
```


Ταυτόχρονα, πρέπει να γίνει έλεγχος εάν ο αντίπαλος βρίσκεται ή όχι στο οπτικό πεδίο του. Επομένως, καλούνται διαδοχικές “if”, οι οποίες ελέγχουν την τιμή όλων των OpponentDist συναρτήσεων – εάν αυτές έχουν την τιμή -0.3, τότε ο Θησέας απέχει απόσταση 3 από τον αντίπαλο, άρα path_array[3] = 3. Εάν έχουν την τιμή -0.5, τότε απέχει απόσταση 2 από τον αντίπαλο, άρα path_array[3] = 2, ενώ, εάν έχουν την τιμή -1, τότε απέχει απόσταση 1 από τον αντίπαλο, άρα path_array[3] = 1. Διαφορετικά, ο Θησέας δεν εντοπίζει τον αντίπαλο μετά τη νέα του κίνηση (path_array[3] = -1):

```
path_array[3] = -1;
if((OpponentDist1(currentPos, opponentPos) == -1) || (OpponentDist3(currentPos,
opponentPos) == -1) || (OpponentDist5(currentPos, opponentPos) == -1) ||
(OpponentDist7(currentPos, opponentPos) == -1) )
    path_array[3] = 1;

if((OpponentDist1(currentPos, opponentPos) == -0.5) || (OpponentDist3(currentPos,
opponentPos) == -0.5) || (OpponentDist5(currentPos, opponentPos) == -0.5) ||
(OpponentDist7(currentPos, opponentPos) == -0.5))
    path_array[3] = 2;

if((OpponentDist1(currentPos, opponentPos) == -0.3) || (OpponentDist3(currentPos,
opponentPos) == -0.3) || (OpponentDist5(currentPos, opponentPos) == -0.3) ||
(OpponentDist7(currentPos, opponentPos) == -0.3))
    path_array[3] = 3;
```

Τέλος, το path ArrayList γεμίζει με τα στοιχεία του path_array:

```
path.add(path_array)
```

οι συνολικές φορές που ο Θησέας έπαιξε (thturns) αυξάνονται κατά ένα:

```
thturns++
```

και επιστρέφεται η νέα θέση του έξυπνου παίκτη.

-Μέθοδος statistics()

Η συνάρτηση αυτή καλείται μία μόνο φορά, στο τέλος του παιχνιδιού, και εκτυπώνει στην οθόνη τα στατιστικά στοιχεία του MinMaxPlayer, για κάθε γύρο που έπαιξε. Για το σκοπό αυτό, καλεί το path ArrayList του συγκεκριμένου γύρου (x) και επιλέγει από αυτό τον πίνακα κάποιο από τα 4 στοιχεία του (y), δηλαδή: path.get(x)[y]. Έτσι, εκτυπώνονται διαδοχικά ο γύρος του Θησέα (οι γύροι που παίζει ο Θησέας είναι περιττοί αριθμοί, δηλαδή 1, 3,

5, 7, 9,...), η κίνησή του (πάνω, δεξιά, κάτω ή αριστερά), μήνυμα συλλογής ενός εφοδίου (σε περίπτωση που ο Θησέας συνέλεξε κάποιο εφόδιο), μήνυμα απόστασης από το κοντινότερο εφόδιο και μήνυμα απόστασης από τον αντίπαλο. Τέλος, εκτυπώνονται οι συνολικές φορές που ο Θησέας κινήθηκε πάνω, οι συνολικές φορές που κινήθηκε δεξιά, οι συνολικές φορές που κινήθηκε κάτω και οι συνολικές φορές που κινήθηκε αριστερά, μέσω των μεταβλητών count1, count3, count5 και count7 που προσθέσαμε στην κλάση MinMaxPlayer.

-Μέθοδος createMySubtree()

Η συνάρτηση αυτή δημιουργεί το πρώτο επίπεδο του δένδρου, που αφορά την αξιολόγηση της πιθανής κίνησης του Θησέα, και έχει τις εξής μεταβλητές:

int currentPos (η θέση του Θησέα)

int opponentCurrentPos (η θέση του Μινώταυρου)

Node root (ο κόμβος-πατέρας του δέντρου)

int depth (το βάθος του κόμβου στο δέντρο του MinMax-αλγόριθμου)

Player b (ο παίκτης Μινώταυρος).

Η συνάρτηση αυτή βρίσκει τις διαθέσιμες κινήσεις του Θησέα. Για κάθε διαθέσιμη κίνηση δημιουργεί έναν εικονικό ταμπλό (Board board1) και ένα αντικείμενο Node που ανήκει στο πρώτο επίπεδο του δέντρου, το οποίο αρχικοποιείται και προστίθεται στο ArrayList children του Node root. Στο σώμα κάθε διαθέσιμης κίνησης του Θησέα καλείται η συνάρτηση createOpponentSubtree() για την ολοκλήρωση του δέντρου. Παρακάτω φαίνεται η δημιουργία ενός Node a1 μόνο για την περίπτωση που ο MinMaxPlayer μπορεί να κινηθεί πάνω (καθώς και όλες οι άλλες περιπτώσεις έχουν την ίδια λογική:

```
if(!this.getBoard().tiles[currentPos].getup())  
{  
    Board board1 = this.getBoard();
```

```

Node a1 = new Node();
a1.nodeBoard = board1;
a1.parent = root;
a1.nodeMove[0] = this.getX();
a1.nodeMove[1] = this.getY() + 1;
a1.nodeMove[2] = 1;
a1.nodeDepth = depth;
a1.nodeEvaluation = evaluate(currentPos, opponentCurrentPos, 1, board1);

createOpponentSubtree(currentPos + board1.getN(), opponentCurrentPos, a1,
depth+1, a1.nodeEvaluation, b);
root.children.add(a1);
}

```

-Μέθοδος createOpponentSubtree()

Η συνάρτηση αυτή δημιουργεί το δεύτερο επίπεδο του δένδρου, που αφορά την αξιολόγηση της πιθανής κίνησης του Μινώταυρου, και έχει τις εξής μεταβλητές:

int currentPos (η θέση του Θησέα)

int opponentCurrentPos (η θέση του Μινώταυρου)

Node parent (ο κόμβος-πατέρας μιας διαθέσιμης κίνησης του Θησέα)

int depth (το βάθος του κόμβου στο δέντρο του MinMax-αλγόριθμου)

Double parentEval (η τιμή της συνάρτησης αξιολόγησης της κίνησης που αντιπροσωπεύει ο συγκεκριμένος κόμβος)

Player a (ο παίκτης Μινώταυρος).

Η συνάρτηση αυτή βρίσκει τις διαθέσιμες κινήσεις του Μινώταυρου. Για κάθε διαθέσιμη κίνηση δημιουργεί έναν εικονικό ταμπλό (Board board1) και ένα αντικείμενο Node που ανήκει στο δεύτερο επίπεδο του δένδρου, το οποίο αρχικοποιείται και προστίθεται στο ArrayList children του Node parent. Παρακάτω φαίνεται η δημιουργία ενός Node a1 μόνο για την περίπτωση που ο Μινώταυρος μπορεί να κινηθεί πάνω (καθώς και όλες οι άλλες περιπτώσεις έχουν την ίδια λογική:

```

if(!this.getBoard().tiles[opponentCurrentPos].getup())
{

```

```
Node a1 = new Node();
a1.nodeBoard = parent.nodeBoard;
a1.parent = parent;
a1.nodeMove[0] = a.getX();
a1.nodeMove[1] = a.getY() + 1;
a1.nodeMove[2] = 1;
a1.nodeDepth = depth;
a1.nodeEvaluation = parentEval - evaluate_minotaur(currentPos,
opponentCurrentPos);
parent.children.add(a1);
}
```