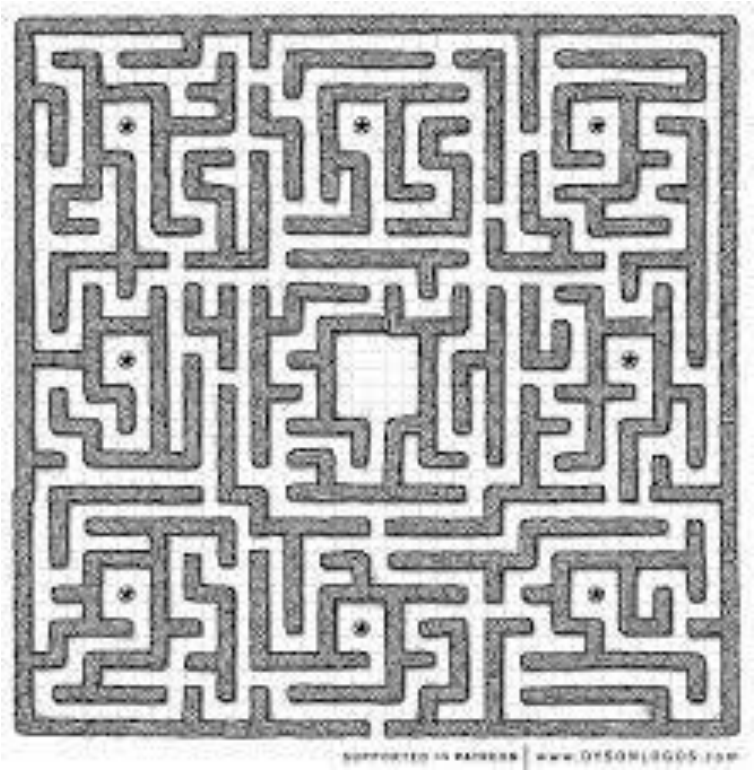
ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Λαβύρινθος

Ο Θησέας και ο Μινώταυρος



Φοιτητές *(ομάδα 128)* :

Βοζίκης Γεώργιος, 9838, vozikisg@ece.auth.gr

Δούμου Ευγενία, 10178, evgedoum@ece.auth.gr

Σιμιτσάκη Ελένη, 10211, simielen@ece.auth.gr

Περιεχόμενα

Λίγα λόγια για το παιχνίδι: 3
Μια σύντομη αναφορά για τις τέσσερις πρώτες κλάσεις (1ο-2ο παραδοτέο) . 4
Κλάση *Supply* 4
Κλάση *Tile* 4
Κλάση *Board* 5
Κλάση *Player* 6
Αναφορά για τις υπόλοιπες κλάσεις (καινούριες ή ανανεωμένες): 7
Κλάση *Node* 7
Κλάση *MinMaxPlayer* 8
Κλάση *Game* 16

Λίγα λόγια για το παιχνίδι:

Ο Θησέας και ο Μινώταυρος είναι ένα παιχνίδι που επινοήθηκε από την ταινία «Μια νύχτα στο μουσείο» η οποία διαδραματίζεται στο Μουσείο Φυσικής Ιστορίας στην Αμερική, στο οποίο τα εκθέματα αποκτούν ζωή για ένα βράδυ και συμμετέχουν σε διάφορες περιπέτειες. Μια από αυτές τις περιπέτειες είναι αυτή του Θησέα και του Μινώταυρου. Ο ατρόμητος Θησέας αποφασίζει να σκοτώσει το Μινώταυρο. Όταν αποβιβάζεται στο νησί, ο Θησέας συναντάει την Αριάδνη, την κόρη του Μίνωα. Η Αριάδνη για να τον βοηθήσει να δραπετεύσει του λέει: «πρέπει να ανακαλύψεις τα διάσπαρτα λάφυρα μέσα στον λαβύρινθο. Μόλις τα μαζέψεις όλα μία καταπακτή θα ανοίξει και θα πέσει μέσα ο Μινώταυρος». Πιο συγκεκριμένα, στο παιχνίδι μας σκοπός του Θησέα είναι να καταφέρει να μαζέψει όλα τα λάφυρα μέσα από τον λαβύρινθο ξεφεύγοντας από τον Μινώταυρο που έχει ως στόχο να τον σκοτώσει. Ο Θησέας έχει ένα φωτόσπαθο, το οποίο φωτίζει το δρόμο του συνεχώς με ένα άσπρο φως. Όταν βρεθεί σε πολύ μικρή απόσταση από ένα λάφυρο, το φωτόσπαθο γίνεται πράσινο. Αντίστοιχα, όταν βρεθεί σε πολύ μικρή απόσταση από τον Μινώταυρο, το φωτόσπαθο γίνεται κόκκινο. Μόλις βρει ο Θησέας όλα τα λάφυρα το παιχνίδι ολοκληρώνεται με νικητή τον Θησέα. Από την άλλη πλευρά, ο Μινώταυρος προσπαθεί να εντοπίσει το Θησέα και να τον σκοτώσει. Ο Μινώταυρος έχει ζωική ικανότητα και μπορεί να διαισθανθεί αν κάποιο από τα εφόδια βρίσκεται σε πολύ μικρή απόσταση από αυτόν. Επίσης μπορεί να μυρίσει το θήραμά του αν αυτό βρεθεί σε μικρή απόσταση, έτσι ώστε να το πλησιάσει να το σκοτώσει. Το παιχνίδι τότε τερματίζει. Τέλος, το παιχνίδι θα τερματίσει και στην περίπτωση που εκτελεστούν 200 ζαριές και δεν έχει βρεθεί ως τότε νικητής, με το παιχνίδι να θεωρείται ισόπαλο, καθώς έχει ξημερώσει και τα εκθέματα πρέπει να πέσουν πάλι σε λήθαργο.

Το παιχνίδι αποτελείται από ένα ταμπλό-λαβύρινθο, μερικά εφόδια-λάφυρα και δύο παίκτες, το Θησέα (MinMaxPlayer) και το Μινώταυρο (Player).Ο Θησέας ξεκινάει από την κάτω αριστερή άκρη (θέση (0,0)), ενώ ο Μινώταυρος από το κέντρο (θέση (N/2,N/2)). Κατά τη διάρκεια του παιχνιδιού κινούνται ανάλογα με τον αριθμό της ζαριάς.Εάν ο Θησέας βρεθεί στο ίδιο πλακίδιο του ταμπλό με το Μινώταυρο, το παιχνίδι τερματίζει με νικητή το Μινώταυρο. Εάν ο Θησέας μαζέψει όλα τα εφόδια, το παιχνίδι τερματίζει με νικητή αυτόν. Σε οποιαδήποτε άλλη περίπτωση το παιχνίδι τερματίζει με ισοπαλία.

Ο αλγόριθμος του προγράμματος για το τρίτο παραδοτέο αναλύεται σε εφτά κλάσεις, σύμφωνα την εκφώνηση.

Μια σύντομη αναφορά για τις τέσσερις πρώτες κλάσεις (1ο-2ο παραδοτέο)

Κλάση *Supply*

Η κλάση αυτή αντιπροσωπεύει τα εφόδια, που υπάρχουν στο παιχνίδι. Παίρνει ως ορίσματα τις ακέραιες μεταβλητές: supplyId, που είναι το id του εφοδίου, τις συντεταγμένες x και y του εφοδίου και τη supplyTileId, που αντιστοιχεί στο id του πλακιδίου, στο οποίο βρίσκεται το εφόδιο στο ταμπλό. Τέλος, σ' αυτή την κλάση φτιάχνουμε τους constructors, τους setters και τους getters των μεταβλητών αυτών, ώστε να αρχικοποιούμε, να τροποποιούμε και να επιστρέφουμε τις τιμές τους.

Κλάση *Tile*

Η κλάση αυτή αντιπροσωπεύει τα πλακίδια, που υπάρχουν στο ταμπλό. Παίρνει ως ορίσματα τις ακέραιες μεταβλητές: tileId, που είναι το id του πλακιδίου, τις συντεταγμένες x και y του πλακιδίου και τέσσερις boolean μεταβλητές up,

down, left και right, οι οποίες δείχνουν εάν υπάρχει τοίχος στην αντίστοιχη πλευρά του πλακιδίου. Τέλος, στην κλάση αυτή φτιάχνουμε τους constructors, τους setters και τους getters των μεταβλητών αυτών, ώστε να αρχικοποιούμε, να τροποποιούμε και να επιστρέφουμε τις τιμές τους.

Κλάση *Board*

Η κλάση Board αντιπροσωπεύει το ταμπλό του παιχνιδιού. Παίρνει ως ορίσματα τις εξής ακέραιες μεταβλητές: N - που είναι ο αριθμός για τις διαστάσεις του ταμπλό (NxN), S - που είναι ο αριθμός των συνολικών εφοδίων που υπάρχουν στο ταμπλό και W - που είναι ο μέγιστος αριθμός των τειχών που μπορούν να προστεθούν στον λαβύρινθο συνολικά. Ακόμα, δηλώνουμε έναν πίνακα tiles με αντικείμενα τύπου Tile και έναν πίνακα supplies με αντικείμενα τύπου Supply. Επιπλέον, φτιάχνουμε τους constructors, τους setters και τους getters των μεταβλητών αυτών, ώστε να αρχικοποιούμε, να τροποποιούμε και να επιστρέφουμε τις τιμές τους. Σε αυτήν την κλάση έχουμε τέσσερις ακόμη μεθόδους:

- Συνάρτηση createTile, η οποία είναι υπεύθυνη για την αρχικοποίηση του πίνακα tiles και την τοποθέτηση των τειχών στο ταμπλό.
- Συνάρτηση createSupply, η οποία είναι υπεύθυνη για τη δημιουργία των εφοδίων και την τοποθέτηση τους στο ταμπλό σε τυχαίες θέσεις (διαφορετικές από τις αρχικές θέσεις των δύο παικτών).
- Συνάρτηση createBoard, στην οποία δημιουργούμε το ταμπλό του παιχνιδιού με ψευδό-τυχαίο τρόπο, καλώντας τις δύο παραπάνω συναρτήσεις (createTile και createSupply).
- Συνάρτηση getStringRepresentation, η οποία είναι υπεύθυνη για τηναπεικόνιση του ταμπλό, των εφοδίων και των παικτών.

Κλάση *Player*

Η κλάση αυτή αντιπροσωπεύει τον παίκτη του παιχνιδιού. Παίρνει ως ορίσματα τις εξής ακέραιες μεταβλητές: playerId, που είναι το id του παίκτη, score, που είναι το συνολικό σκορ του κάθε παίκτη, το οποίο αυξάνεται κατά την εύρεση εφοδίων και οι συντεταγμένες x και y. Επίσης, δηλώνεται μια μεταβλητή name τύπου String, που αντιστοιχεί στο όνομα του παίκτη και μια μεταβλητή board τύπου Board, που αντιστοιχεί στο ταμπλό του παιχνιδιού. Όπως και στις άλλες κλάσεις έτσι και εδώ έχουμε τους constructors, setters και getters των μεταβλητών. Επίσης σε αυτή την κλάση έχουμε δύο ακόμη μεθόδους:

- ***public int randomDice():*** Στην συνάρτηση αυτή δηλώνουμε έναν πίνακα dice με τέσσερις τιμές (1,3,5,7), δηλαδή τις τιμές που μπορεί να πάρει το ζάρι. Στη συνέχεια, δηλώνουμε μία μεταβλητή randDice, στην οποία δίνουμε μια τυχαία τιμή από το 0 έως και το 3 η οποία αναφέρεται στις θέσεις (index) του πίνακα. Επιστρέφουμε την τιμή του πίνακα που είναι αποθηκευμένη στη θέση randDice και με τον τρόπο αυτόν παίρνουμε με τυχαίο τρόπο την κίνηση ενός παίκτη.
- ***public int[] move(int id, int dice):*** Η συνάρτηση αυτή είναι υπεύθυνη για την κίνηση των παικτών πάνω στο ταμπλό. Η συνάρτηση αυτή παίρνει ως ορίσματα τον αριθμό του πλακιδίου στο οποίο βρίσκεται ο παίκτης (id) και τον αριθμό της ζαριάς (dice), ο οποίος μπορεί να είναι είτε τυχαίος είτε όχι. Αρχικά, δηλώνουμε έναν πίνακα array στον οποίο θα αποθηκεύουμε μετά από κάθε κίνηση τις συντεταγμένες και το id του παίκτη και το id του εφοδίου που έχει μαζέψει (εάν έχει πάρει εφόδιο). Στη συνέχεια, χρησιμοποιούμε μια switch, με όρισμα τη ζαριά του παίκτη (1-πάνω, 3-δεξιά, 5-κάτω, 7-αριστερά). Σε κάθε τιμή ζαριάς γίνονται τα ακόλουθα:

- Στην περίπτωση που ο παίκτης είναι ο Μινώταυρος, αρχικά ελέγχεται εάν υπάρχει τείχος που εμποδίζει την κίνηση που αντιστοιχεί στην ζαριά του case. Εάν υπάρχει τείχος, εμφανίζεται ανάλογο μήνυμα (για τον Θησέα αντίστοιχος έλεγχος επιτρεπόμενης κίνησης προς τη συγκεκριμένη κατεύθυνση έχει προηγηθεί στη συνάρτηση getNextMove της κλάσης MinMaxPlayer για την επιλογή της βέλτιστης κίνησης).
- Στη συνέχεια ακολουθεί η ανανέωση των συντεταγμένων και του tileId του παίκτη (είτε του Θησέα είτε του Μινώταυρου εφόσον μπορεί να κινηθεί) κατά αντιστοιχία της κίνησης. Οι νέες αυτές τιμές αποθηκεύονται στη συνέχεια στον πίνακα array.
- Τέλος στην περίπτωση του Θησέα, ελέγχεται αν μάζεψε κάποιο εφόδιο, οπότε και αυξάνεται το σκορ του κατά 1 και αποθηκεύεται η θέση (supplyTileId) του εφοδίου στην τελευταία θέση του πίνακα array.

Στο τέλος της συνάρτησης επιστρέφεται ο πίνακας array.

Αναφορά για τις υπόλοιπες κλάσεις (καινούριες ή ανανεωμένες):

Κλάση *Node*

Η κλάση αυτή αντιπροσωπεύει τη δημιουργία του κόμβου. Παίρνει ως ορίσματα τις: Node parent, ο κόμβος-πατέρας του κόμβου που δημιουργήσαμε, ArrayList<Node> children, ο δυναμικός πίνακας που περιλαμβάνει τα παιδιά του κόμβου που δημιουργήσαμε, int nodeDepth, το βάθος του κόμβου στο δέντρο του MinMax Αλγορίθμου, int[] nodeMove, η κίνηση που αντιπροσωπεύει το Node, σαν πίνακας ακεραίων που περιλαμβάνει το x, y του πλακιδίου της τρέχουσας θέσης (πριν εκτελεστεί η κίνηση) και τον αριθμό του ζαριού, Board nodeBoard, το ταμπλό του παιχνιδιού για το συγκεκριμένο

κόμβο-κίνηση και double nodeEvaluation, η τιμή της συνάρτησης αξιολόγησης της κίνησης που αντιπροσωπεύει ο συγκεκριμένος κόμβος. Τέλος, σ' αυτή την κλάση φτιάχνουμε τους constructors, τους setters και τους getters των μεταβλητών αυτών, ώστε να αρχικοποιούμε, να τροποποιούμε και να επιστρέφουμε τις τιμές τους.

Κλάση *MinMaxPlayer*

Η κλάση αυτή αντιπροσωπεύει τον παίκτη που παίζει χρησιμοποιώντας τον αλγόριθμο MinMax (Θησέα), η οποία κληρονομεί την κλάση Player και παίρνει ως ορίσματα την εξής επιπλέον μεταβλητή: ArrayList <Integer[]> path, η οποία περιλαμβάνει πληροφορίες για το ζάρι, για το αν πήρε εφόδιο ή όχι ο Θησέας, την απόστασή του από ένα εφόδιο και τον αντίπαλο (εφόσον μπορεί να τα δει). Όπως και στις άλλες κλάσεις έτσι και εδώ έχουμε τους constructors, setters και getters των μεταβλητών. Σε αυτή την κλάση έχουμε εφτά ακόμα μεθόδους:

- ***public double f (double nearSupplies, double opponentDist):*** η οποία είναι η συνάρτηση στόχου. Πιο αναλυτικά, η συνάρτηση αυτή δέχεται ως ορίσματα δύο μεταβλητές (nearSupplies και opponentDist) που αντιστοιχούν στην απόσταση του παίκτη από το κοντινότερο εφόδιο και στην απόστασή του από τον αντίπαλο (εφόσον είναι σε θέση που μπορεί να τα δει). Η συνάρτηση αυτή χρησιμοποιεί την σχέση (nearSupplies * 0.46) + (opponentDist * 0.54) η οποία επιστρέφει μια τιμή που αντιστοιχεί στην αξιολόγηση της συγκεκριμένης κίνησης που εξετάζεται κάθε φορά (η κλάση αυτή καλείται μέσα στη συνάρτηση evaluate της ίδιας κλάσης με στόχο την αξιολόγηση της βέλτιστης κίνησης).
- ***public double evaluate(int currentPos, int minotaurPos, int dice):*** η οποία αξιολογεί την κίνηση του παίκτη, όταν αυτός έχει τη ζαριά (dice), δεδομένου ότι βρίσκεται στη θέση currentPos. Η συνάρτηση επιστρέφει

την αξιολόγηση της κίνησης, σύμφωνα με τη συνάρτηση στόχου που ορίσαμε παραπάνω. Αρχικά, αρχικοποιούμε με 0 τις μεταβλητές nearSupplies, opponentDist και δηλώνουμε μία ακόμα, την temporaryPos, η οποία θα μας βοηθάει στο να κινήσουμε τον παίκτη θεωρητικά πάνω στο ταμπλό. Με τη χρήση της switch, ανάλογα με τον αριθμό της ζαριάς (1,3,5,7) αξιολογούμε την κάθε κίνηση ακολουθώντας τα εξής βήματα:

Σε κάθε case, ελέγχουμε αρχικά αν ο παίκτης είναι ο Θησέας με μια if (αν id==1 που αντιστοιχεί στον παίκτη Θησέα όπως ορίστηκε στην κλάση game στην συνάρτηση main) και με τη χρήση μιας for αποκτούμε πρόσβαση σε όλα τα εφόδια του πίνακα supplies[] από το πρώτο έως το τελευταίο στοιχείο του, έτσι ώστε να γνωρίζουμε το tileId του κάθε εφοδίου και να το συγκρίνουμε με αυτό του παίκτη. Σε κάθε επανάληψη της for:

Αρχικά ελέγχουμε αν στην currentPos που βρίσκεται ο παίκτης και ανάλογα με τον αριθμό της ζαριάς από το αντίστοιχο case, ελέγχουμε αν από εκείνη τη θέση μπορεί να κινηθεί στην αντίστοιχη κατεύθυνση, δηλαδή αν δεν υπάρχει τείχος, τότε αυτή η κίνηση είναι εφικτή και τότε θέτουμε στην temporaryPos το αθροισμα της currentPos με την αντίστοιχη κίνηση προς εκείνη την κατεύθυνση.

Στη νέα αυτή θέση (temporaryPos η οποία απέχει 1 tile από το currentPos) ελέγχουμε αν υπάρχει εφόδιο.

Αν υπάρχει προσθέτουμε στην μεταβλητή nearSupplies την τιμή 1 και προχωρούμε στην επόμενη επανάληψη της for (καθώς κάθε εφόδιο βρίσκεται σε ένα και μόνο tile).

Εάν όμως δεν βρεθεί εφόδιο, η τιμή της μεταβλητής temporaryPos ανανεώνεται ξανά κατά ένα tile προς την ίδια κατεύθυνση που εξετάζουμε (εφόσον αρχικά ελέγξουμε αν είναι δυνατή αυτή η κίνηση) και ελέγχεται εκ νέου εάν στη θέση αυτή (temporaryPos η οποία τώρα απέχει 2 tile από το currentPos) βρίσκεται εφόδιο ή όχι.

Εάν βρεθεί εφόδιο, τότε στη μεταβλητή nearSupplies προστίθεται η τιμή 0.5 και προχωρούμε στην επόμενη επανάληψη της for.

Εάν όμως δεν βρεθεί εφόδιο, η τιμή της μεταβλητής temporaryPos ανανεώνεται ξανά κατά μια κίνηση προς την ίδια κατεύθυνση(εφόσον αρχικά ελέγξουμε αν είναι δυνατή αυτή η κίνηση) και ελέγχεται εκ νέου εάν στη θέση αυτή (temporaryPos η οποία τώρα απέχει 3 tile από το currentPos) βρίσκεται εφόδιο ή όχι.

Εάν βρεθεί εφόδιο, τότε στη μεταβλητή nearSupplies προστίθεται η τιμή 0.3.

Η διαδικασία αυτή επαναλαμβάνεται για όλα τα στοιχεία του πίνακα supplies και σε κάθε tile που βρίσκεται ο Θησέας (σε θέση temporaryPos) ταυτόχρονα με κάποιο εφόδιο, η μεταβλητή nearSupplies ανανεώνεται με τον εξής τρόπο:

- Εάν ο Θησέας βρίσκεται σε απόσταση(από το currentPos) ενός πλακιδίου από κάποιο εφόδιο, τότε η μεταβλητή nearSupplies αυξάνεται κατά 1.
- Εάν ο Θησέας βρίσκεται σε απόσταση(από το currentPos) δύο πλακιδίων από κάποιο εφόδιο, τότε η μεταβλητή nearSupplies αυξάνεται κατά 0.5.

- Εάν ο Θησέας βρίσκεται σε απόσταση(από το currentPos) τριών πλακιδίων από κάποιο εφόδιο, τότε η μεταβλητή nearSupplies αυξάνεται κατά 0.3.

  (η τιμή κάθε φορά αυξάνεται διότι θέλουμε μια κίνηση να κρίνεται θετικότερη αν υπάρχουν 3 εφόδια στη σειρά προς αυτή την κατεύθυνση από μία άλλη που υπάρχουν 2 ή 1 εφόδια μόνο).

Σε οποιαδήποτε άλλη περίπτωση ο Θησέας δεν μπορεί να δει κάποιο εφόδιο, δηλαδή αν αυτό βρίσκεται σε απόσταση μεγαλύτερη των 3 tiles από το currentPos (άρα η μεταβλητή nearSupplies θα πάρει την τιμή 0).

Με τον ίδιο τρόπο εργαζόμαστε και για την απόσταση από τον αντίπαλο παίκτη (είτε αυτός είναι ο Θησέας είτε ο Μινώταυρος), απλά με αντίστοιχες αρνητικές τιμές. Δηλαδή:

- Εάν ο παίκτης βρίσκεται σε απόσταση ενός πλακιδίου από τον αντίπαλο παίκτη, τότε η μεταβλητή opponentDist παίρνει την τιμή -1.
- Εάν ο παίκτης βρίσκεται σε απόσταση δύο πλακιδίων από τον αντίπαλο παίκτη, τότε η μεταβλητή opponentDist παίρνει την τιμή -0.5.
- Εάν ο παίκτης βρίσκεται σε απόσταση τριών πλακιδίων από τον αντίπαλο παίκτη, τότε η μεταβλητή opponentDist παίρνει την τιμή -0.3.

Η διαδικασία αυτή είναι ίδια είτε ελέγχεται η απόσταση Θησέα-Μινώταυρου είτε η απόσταση Μινώταυρου-Θησέα για όλες τις τιμές του ζαριού (με μόνη διαφορά στην ανανέωση της μεταβλητής temporaryPos ανάλογα με την κατεύθυνση της κίνησης).

Τέλος, καλούμε την συνάρτησης στόχου (f) με ορίσματα τις τιμές που προκύπτουν από το εσωτερικό των ελέγχων και επιστρέφουμε την τιμή που επιστρέφεται από την συνάρτηση στόχου.

- ***public int[] getNextMove(int currentPos, int opponentCurrentPos):*** η οποία είναι υπεύθυνη για την επιλογή της τελικής κίνησης του παίκτη. Στον κώδικα μας αρχικά δηλώνουμε ένα αντικείμενο root τύπου Node και καλώντας την createMySubtree δημιουργούμε τη ρίζα. Επίσης, δηλώνουμε μια μεταβλητή dice, η οποία παίρνει τιμή την τιμή που επιστρέφει η συνάρτηση chooseMinMaxMove και έναν πίνακα move, ο οποίος αρχικοποιείται από τον πίνακα που επιστρέφεται από την κλάση της συνάρτησης move (κλάσης Player).

  Έπειτα, με τη χρήση της μεταβλητής sup ελέγχουμε εάν ο Θησέας στην καινούρια του θέση πήρε ή όχι κάποιο εφόδιο, δηλαδή αν η τιμή της μεταβλητής sup είναι 0, σημαίνει ότι ο Θησέας δεν πήρε εφόδιο, ενώ αν είναι 1 σημαίνει ότι πήρε.

  Στη συνέχεια, ελέγχουμε αν στην νέα θέση του Θησέα υπάρχει εφόδιο.

  Δηλώνουμε δύο καινούργιες μεταβλητές, την maxS, όπου αποθηκεύουμε σε κάθε έλεγχο μιας κατεύθυνσης την μικρότερη απόσταση από ένα εφόδιο - εάν υπάρχει - και την supplieDistance, η οποία αντιπροσωπεύει την τελική κοντινότερη απόσταση από ένα εφόδιο στην νέα θέση του παίκτη.

  Στην νέα θέση πλέον του Θησέα, ελέγχουμε την απόσταση από το κοντινότερο εφόδιο (αν υπάρχει) για κάθε μία από τις πιθανές κατευθύνσεις.

  Ο έλεγχος αυτός γίνεται με παρόμοιο τρόπο με την λογική της συνάρτησης evaluate(), με τη διαφορά πως αν το εφόδιο απέχει από το currentPos 1 tile, αποθηκεύουμε στην μεταβλητή maxS την τιμή 1, αν

απέχει 2 tiles αποθηκεύουμε την τιμή 2, ενώ αν απέχει 3 tiles αποθηκεύουμε την τιμή 3.

Ελέγχονται διαδοχικά και οι 4 πιθανές κατευθύνσεις. Μετά τον έλεγχο της πρώτης κατεύθυνσης θέτουμε supplieDistance = maxS. Σε κάθε επόμενο έλεγχο έχουμε τις εξής περιπτώσεις:

1. Εάν το supplieDistance έχει τιμή διάφορη του 0 (δηλαδή σε προηγούμενους ελέγχους είχε βρεθεί εφόδιο σε κάποια απόσταση) τότε για να θέσουμε το supplieDistance ίσο με το maxS της συγκεκριμένης κατεύθυνσης που ελέγξαμε πρέπει το maxS < supplieDistance (δηλαδή το εφόδιο να βρίσκεται πιο κοντά).
2. Εάν το supplieDistance έχει τιμή 0 (δηλαδή σε προηγούμενους ελέγχους δεν είχε βρεθεί εφόδιο σε απόσταση 3 tiles από τη θέση του Θησέα) τότε για να θέσουμε το supplieDistance ίσο με το maxS της συγκεκριμένης κατεύθυνσης που βρισκόμαστε πρέπει το maxS > supplieDistance (δηλαδή να έχει βρεθεί κάποιο εφόδιο σε απόσταση 3 tiles από τη θέση του Θησέα).

Εργαζόμαστε με αντίστοιχο τρόπο για την απόσταση (minotaurDistance) από τον Μινώταυρο. Στο τέλος, ανανεώνουμε την μεταβλητή path και επιστρέφουμε τον πίνακα move.

- ***void createMySubTree(Node root, int currentPos, int opponentCurrentPos, int depth):*** στην οποία αρχικά δηλώνουμε την μεταβλητή cloneBoard, με την οποία δημιουργούμε έναν κλώνο του πίνακα και την parentEval, στην οποία αποθηκεύουμε την τιμή της αξιολόγησης της πιθανής κίνησης. Στη συνέχεια, ελέγχουμε αν η κίνηση προς τα πάνω είναι πιθανή. Πιο συγκεκριμένα, εάν προς τα πάνω δεν υπάρχει τείχος, τότε δηλώνουμε έναν πίνακα MoveUp τριών θέσεων και

τον αρχικοποούμε με τα δεδομένα της κίνησης του κόμβου, δηλαδή τις συντεταγμένες του παίκτη πριν την κίνηση και τον αριθμό της ζαριάς (1). Έτσι, αξιολογούμε την κίνηση, δημιουργούμε κόμβο για την κίνηση και τον προσθέτουμε στη ρίζα και τέλος καλούμε την createOpponentSubtree.

Αντίστοιχα εργαζόμαστε για τις κινήσεις προς τα δεξιά, κάτω και αριστερά.

- ***void createOpponentSubtree(int currentPos, int opponentCurrentPos, Node parent, int depth, double parentEval):*** στην οποία αρχικά δηλώνουμε την μεταβλητή cloneBoard, με την οποία δημιουργούμε έναν κλώνο του πίνακα και την Eval, στην οποία αποθηκεύουμε το άθροισμα της αξιολόγησης της συγκεκριμένης κίνησης με την τιμή της αξιολόγησης της κίνησης-γονέα αυτής. Στη συνέχεια, ελέγχουμε αν η κίνηση προς τα πάνω είναι πιθανή. Πιο συγκεκριμένα, εάν προς τα πάνω δεν υπάρχει τείχος, τότε δηλώνουμε έναν πίνακα MoveUp τριών θέσεων και τον αρχικοποιούμε με τα δεδομένα της κίνησης του κόμβου, δηλαδή τις συντεταγμένες του παίκτη πριν την κίνηση και τον αριθμό της ζαριάς (1). Έτσι, αξιολογούμε την κίνηση, αποθηκεύουμε το άθροισμα αυτής με την parentEval στη μεταβλητή Eval, δημιουργούμε κόμβο για την κίνηση και τον προσθέτουμε στον κόμβο-πατέρα.

  Αντίστοιχα εργαζόμαστε για τις κινήσεις προς τα δεξιά, κάτω και αριστερά.
- ***int chooseMinMaxMove(Node root):*** η οποία υλοποιεί τον MinMax αλγόριθμο για να βρει τη βέλτιστη διαθέσιμη κίνηση και επιστρέφει το δείκτη της καλύτερης κίνησης.

  Αρχικά, δημιουργούμε Random αριθμούς. Στη συνέχεια, θέλουμε να βρούμε, για κάθε κλαδί του δέντρου, τη μικρότερη τιμή των φύλλων του,

οπότε με τη χρήση μιας for για τα κλαδιά του δέντρου σε κάθε επανάληψη της θέτουμε στην αρχή στη μεταβλητή min, η οποία αντιστοιχεί στην μικρότερη τιμή αξιολόγησης, την τιμή της αξιολόγησης του πρώτου φύλλου του κλαδιού από τα αριστερά. Στη συνέχεια, με μία δεύτερη for ελέγχουμε και τις τιμές αξιολόγησης των υπόλοιπων φύλλων και κάθε φορά ελέγχουμε με δυο if, στη πρώτη περίπτωση αν η τιμή της αξιολόγησης του φύλλου που βρισκόμαστε είναι μικρότερη από τη τιμή που έχουμε θέσει στο min και τότε την αντικαθιστούμε, ενώ με τη δεύτερη if ελέγχουμε αν η τιμή αξιολόγησης του συγκεκριμένου φύλλου είναι ίση με το min, ώστε να θέσουμε ως min μία από τις δύο τυχαία. Εργαζόμαστε ανάλογα και για την μεγαλύτερη αξία.

- ***public void statistics (int n):*** η οποία εκτυπώνει στοιχεία για τις κινήσεις του παίκτη σε κάθε γύρο του παιχνιδιού, δηλαδή το ζάρι που επέλεξε και τις ενέργειες που έκανε, καθώς και στατιστικά στοιχεία για το σύνολο των κινήσεών του για όλους τους γύρους συνολικά. Η συνάρτηση αυτή δέχεται ως όρισμα έναν ακέραιο αριθμό n ο οποίος αντιστοιχεί στον συνολικό αριθμό γύρων που εκτελέστηκαν σε κάθε παιχνίδι.

  Συγκεκριμένα, χρειάζεται να τυπώνονται:

  - Ο αριθμός των εφοδίων που έχει μαζέψει ο Θησέας.
  - Η απόστασή του Θησέα από το κοντινότερο εφόδιο. Εφόσον μπορεί να το δει, δηλαδή δεν υπάρχουν τείχη μεταξύ.
  - Η απόστασή του Θησέα από τον αντίπαλο. Εφόσον μπορεί να τον δει, δηλαδή δεν υπάρχουν τείχη μεταξύ.

  και μία φορά:

  - Οι φορές που ο παίκτης επέλεξε να κινηθεί μπροστά (1).
  - Οι φορές που ο παίκτης επέλεξε να κινηθεί δεξιά (3).
  - Οι φορές που ο παίκτης επέλεξε να κινηθεί πίσω (5).

- Οι φορές που ο παίκτης επέλεξε να κινηθεί αριστερά (7).

Στον κώδικα μας, αρχικά, με χρήση μιας επανάληψης, τυπώνουμε τον τρέχοντα γύρο και την τιμή της ζαριάς που επέλεξε ο Θησέας. Ταυτόχρονα, χρησιμοποιούμε τέσσερις μεταβλητές (up, down, right, left), τις οποίες αυξάνουμε κατά 1 κάθε φορά που ο Θησέας διαλέγει την αντίστοιχη κίνηση, ώστε να μετρήσουμε συνολικά πόσες φορές κινήθηκε μπροστά, πίσω, δεξιά ή αριστερά. Στη συνέχεια, ελέγχουμε αν ο Θησέας, με την τρέχων κίνησή του, πήρε κάποιο εφόδιο και τότε τυπώνουμε μήνυμα ότι πήρε εφόδιο και το συνολικό αριθμό εφοδίων που έχει στην κατοχή του.

Επίσης, ελέγχουμε αν υπάρχει ή όχι κάποιο εφόδιο ή ο αντίπαλος του κοντά του (μέγιστη απόσταση 3) και τυπώνουμε ανάλογο μήνυμα.

Η ίδια διαδικασία επαναλαμβάνεται για όλους τους γύρους. Στον τελευταίο όμως γύρο, γίνεται μια επιπλέον εκτύπωση για το σύνολο των κινήσεών του Θησέα για όλους τους γύρους συνολικά(up, down, right και left).

Κλάση *Game*

Σε αυτήν την τελευταία κλάση του προγράμματος, έχουμε ως όρισμα μία ακέραια μεταβλητή μόνο, τη round, που αντιστοιχεί στον τρέχον γύρο του παιχνιδιού. Όπως και στις προηγούμενες κλάσεις, έτσι και εδώ έχουμε τους constructors, τους setters και τους getters της μεταβλητής. Τέλος, σε αυτή την κλάση έχουμε και τη βασική συνάρτηση του προγράμματος, main, η οποία είναι υπεύθυνη για τη δημιουργία, την ορθή εξέλιξη και την λήξη του παιχνιδιού.

- ***public static void main(String[] args):*** Αρχικά, δηλώνουμε μεταβλητές και τις αρχικοποιούμε σύμφωνα με τα ζητούμενα των οδηγιών (N=15, η

διάσταση του ταμπλό, S=4, τα συνολικά εφόδια, W=(3*N*N+1)/2, τα συνολικά τείχη, n=100, για τις ζαριές των παικτών οι οποίες είναι 2n, i=0, μία μεταβλητή που αντιπροσωπεύει το γύρο μέσα στις επαναλήψεις και μία Boolean μεταβλητή, η οποία τερματίζει τον βρόγχο do...while που χρησιμοποιούμε. Στη συνέχεια δημιουργείται το ταμπλό με τα αντίστοιχα ορίσματα N,S,W και οι 2 παίκτες (Θησέας τύπου MinMaxPlayer και Μινώταυρος τύπου Player). Γίνεται μια εκτύπωση του ταμπλό για την αρχική του κατάσταση, πριν δηλαδή οι παίκτες αρχίσουν να ρίχνουν το ζάρι και να κινούνται μέσα στο ταμπλό. Η μεταβλητή i, που έχει αρχικοποιηθεί με 0, αυξάνεται κατά 1 μέσα στη δομή επανάληψης, έτσι ώστε να αντιπροσωπεύει τον τρέχων γύρο του παιχνιδιού. Οι εντολές που εκτελούνται σε κάθε επανάληψη του βρόγχου do...while είναι οι εξής:

- Ο παίκτης με id=2 (Μινώταυρος) παίζει (καλείται η randomDice για τυχαία κίνηση).
- Έλεγχος αν ο παίκτης που έπαιξε μόλις είναι ο νικητής, ώστε να τερματιστεί το παιχνίδι με νικητή τον Μινώταυρο (αν βρέθηκε σε ίδιο tile με Θησέα).
- Ο παίκτης με id=1 (Θησέας) παίζει (βέλτιστη κίνηση finalDice).
- Έλεγχος αν ο παίκτης που έπαιξε μόλις είναι ο νικητής, ώστε να τερματιστεί το παιχνίδι με νικητή τον Θησέα (εάν μάζεψε όλα τα εφόδια του ταμπλό).
- Αφού έχουν παίξει και οι δύο παίκτες στον τρέχων γύρο, γίνεται έλεγχος για το αν έχουν βρεθεί στο ίδιο πλακίδιο. Αν ισχύει αυτό, τότε το παιχνίδι τερματίζει με νικητή τον Μινώταυρο.
- Έλεγχος αν έχουν παιχτεί 2n ζαριές. Αν ισχύει αυτό, τότε το παιχνίδι τερματίζει με ισοπαλία.

Για όλα τα παραπάνω, τυπώνονται σχετικά μηνύματα, δηλαδή σε κάθε γύρο τυπώνονται ο τρέχον γύρος, η κίνηση του κάθε παίκτη, το ταμπλό με τις θέσεις των παικτών και των εφοδίων. Τέλος, τυπώνεται το τελικό ταμπλό και τα στατιστικά για τον κάθε γύρο (με την κλήση της συνάρτησης statistics της κλάσης MinMaxPlayer).