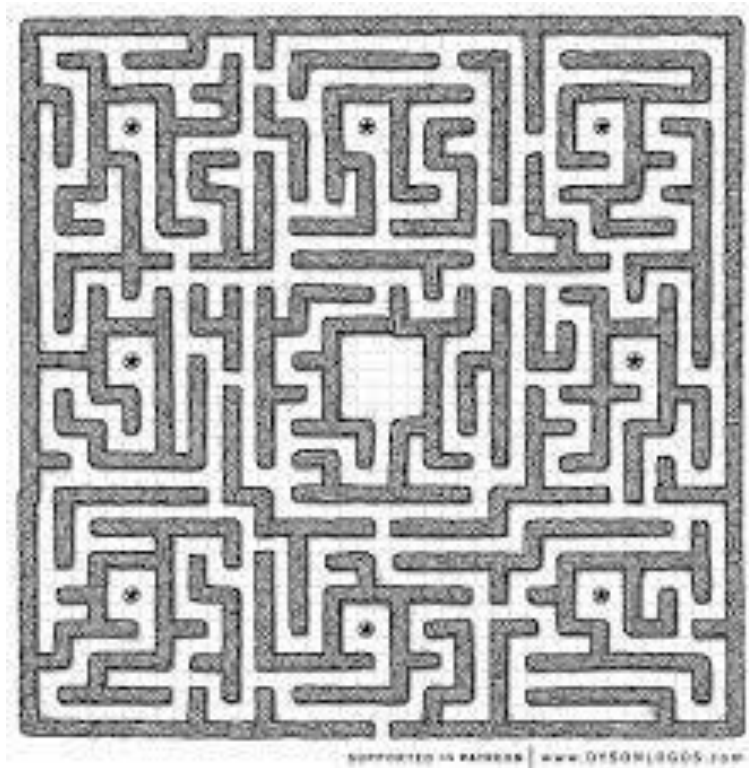


# ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Λαβύρινθος

Ο Θησέας και ο Μινώταυρος



**Φοιτητές (ομάδα 128):**

Βοζίκης Γεώργιος, 9838, [vozikisg@ece.auth.gr](mailto:vozikisg@ece.auth.gr)

Δούμου Ευγενία, 10178, [evgedoum@ece.auth.gr](mailto:evgedoum@ece.auth.gr)

Σιμιτσάκη Ελένη, 10211, [simielen@ece.auth.gr](mailto:simielen@ece.auth.gr)

## Περιεχόμενα

<b>Λίγα λόγια για το παιχνίδι:</b> .....	3
<b>Μια σύντομη αναφορά για τις τρεις πρώτες κλάσεις (πρώτο παραδοτέο):</b> .....	4
Κλάση Supply:.....	4
Κλάση Tile .....	4
Κλάση Board: .....	5
<b>Αναφορά για τις υπόλοιπες κλάσεις (καινούριες ή ανανεωμένες):</b> .	6
Κλάση Player: .....	6
Κλάση HeuristicPlayer: .....	7
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 (HeuristicPlayer) 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 δεύτερο παραδοτέο αναλύεται σε έξι κλάσεις, σύμφωνα την εκφώνηση.

### **Μια σύντομη αναφορά για τις τρεις πρώτες κλάσεις (πρώτο 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 ( διάστασης  $N \times N$ ), *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 την απεικόνιση του ταμπλό, των εφοδίων και των παικτών.

### **Αναφορά για τις υπόλοιπες κλάσεις (καινούριες ή ανανεωμένες):**

#### **Κλάση Player:**

Η κλάση αυτή αντιπροσωπεύει τον παίκτη του παιχνιδιού που κινείται με τυχαίο τρόπο στο ταμπλό. Παίρνει ως ορίσματα τις εξής αμέριστε μεταβλητές: *playerId*-που είναι το *id* του παίκτη, *score*-που είναι το συνολικό σκορ του κάθε παίκτη, το οποίο αυξάνεται κατά την εύρεση εφοδίων (αναφέρεται μόνο στον Θεσέα) 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*, τους *setters* και τους *getters* των μεταβλητών. Επίσης σε αυτή την κλάση έχουμε δύο ακόμη μεθόδους:

- ***Public int randomDice():*** Στην συνάρτηση αυτή δηλώνουμε έναν πίνακα *dice* και τον αρχικοποιούμε με τις τιμές (1,3,5,7), δηλαδή με 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 *HeuristicPlayer* 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*.

### Κλάση *HeuristicPlayer*:

Η κλάση αυτή αντιπροσωπεύει τον παίκτη που παίζει με στρατηγική (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, τον αριθμό των πόντων που προσέφερε η κίνηση του παίκτη και αν βρίσκεται κοντά σε ένα αντικείμενο, καθώς και αν βρίσκεται κοντά στον αντίπαλό του. Όπως και στις άλλες κλάσεις έτσι και εδώ έχουμε τους *constructors* της μεταβλητής *path*. Σε αυτή την κλάση έχουμε τέσσερις 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 Μινώταυρο (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 (Θησέα) 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, 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 θέτουμε στην μεταβλητή *temporaryPos* την τιμή της νέας θέσης του παίκτη εφόσον πραγματοποιήσει θεωρητικά την κίνηση της ζαριάς που ελέγχουμε στο συγκεκριμένο case (δεδομένου ότι έχουμε ήδη ελέγξει στην *getNextMove()* ότι στη *currentPos* ο Θησέας μπορεί να κινηθεί προς τη συγκεκριμένη κατεύθυνση, δηλαδή δεν υπάρχει τείχος που να τον εμποδίζει). 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 ελέγχουμε 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 το Μινώταυρο, απλά με αντίστοιχες αρνητικές τιμές. Δηλαδή:

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

Η διαδικασία αυτή είναι ίδια 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 minotaurPos):** η οποία είναι υπεύθυνη για την επιλογή της τελικής κίνησης του παίκτη. Στον κώδικα μας αρχικά δηλώνουμε μια μεταβλητή *maxMoves* με την οποία μετράμε τον μέγιστο αριθμό των πιθανών κινήσεων (δηλαδή αν δεν υπάρχει τείχος προς μία κάποια κατεύθυνση, η κίνηση προς αυτή θεωρείται πιθανή). Ταυτόχρονα, δεσμεύουμε δυναμικά έναν δισδιάστατο πίνακα *possibleMoves*, ο οποίος αποτελείται από 2 γραμμές και *maxMoves* στήλες (όσες και οι πιθανές κινήσεις) ώστε να αποθηκεύουμε την πιθανή κίνηση (στην πρώτη γραμμή) με την αντίστοιχη αξιολόγησή της (στη δεύτερη γραμμή).

Με μία δομή επανάληψης *do-while* τοποθετούμε στον πίνακα τις πιθανές κινήσεις και την αξιολόγηση αυτών με τη χρήση μιας μεταβλητής *i*, η οποία αυξάνεται κάθε φορά που εντοπίζεται και τοποθετείται στον πίνακα μια κίνηση.

Στη συνέχεια ταξινομούμε τον πίνακα κατά φθίνουσα τιμή αξιολόγησης (σύμφωνα με την δεύτερη δηλαδή γραμμή).

Ακόμη, δηλώνουμε δύο μεταβλητές, την *finalDice*, που αντιστοιχεί στην τελική κίνηση του Θησέα και την *sameValues*, που μας δείχνει πόσες πιθανές κινήσεις έχουν την ίδια μέγιστη αξιολόγηση (*sameValues*=1 σημαίνει ότι υπάρχει μοναδική κίνηση με την μέγιστη αξιολόγηση).

Με μία δομή επανάληψης *for* συγκρίνουμε στον ταξινομημένο κατά φθίνουσα σειρά πίνακα *possibleMoves*, την πρώτη αξιολόγηση (που είναι και η μεγαλύτερη) με όλες τις υπόλοιπες. Κάθε φορά που αυτή η μέγιστη αξιολόγηση είναι ίση με κάποια άλλη, αυξάνουμε τον μετρητή *sameValues*.

Στο τέλος της *for*, η *sameValues* μπορεί να έχει τις εξής τιμές:

- ❖ 4, δηλαδή οι πιθανές κινήσεις είναι τέσσερις και όλες με την ίδια αξιολόγηση (μέγιστη)
- ❖ 3, δηλαδή τρεις από τις πιθανές κινήσεις έχουν ίδια αξιολόγηση (μέγιστη)
- ❖ 2, δηλαδή δύο από τις πιθανές κινήσεις έχουν την ίδια αξιολόγηση (μέγιστη)
- ❖ 1, δηλαδή η μέγιστη αξιολόγηση αντιστοιχεί σε μία και μοναδική κίνηση.

Έτσι, εάν βρισκόμαστε σε κάποια από τις 3 πρώτες τιμές της *sameValues* (4,3,2) δημιουργούμε ένα πίνακα *randomMoves* στον οποίο αποθηκεύουμε όλες τις πιθανές κινήσεις με ίδια αξιολόγηση (μέγιστη)

και θέτοντας σε μια μεταβλητή *randMove* τυχαίες τιμές από το 0 έως και την τιμή της τελευταίας θέσης του πίνακα, επιστρέφουμε στην μεταβλητή *finalDice* το περιεχόμενο του πίνακα *randomMoves* στη θέση *randMove* (με τον τρόπο αυτό επιλέγουμε τυχαία μια κίνηση με μέγιστη αξιολόγηση).

Αν βρισκόμαστε στην τελευταία περίπτωση (1), θέτουμε στην μεταβλητή *finalDice* την μοναδική κίνηση που αντιστοιχεί στη μέγιστη αξιολόγηση (αυτήν που βρίσκεται στην πρώτη θέση του πίνακα *possibleMoves*).

Αφού έχουμε επιλέξει το *finalDice*, κινούμε τον παίκτη ανάλογα με την τιμή της επιλογής (1:μπροστά, 3:δεξιά, 5:πίσω, 7:αριστερά) καλώντας τη συνάρτηση *move()* της κλάσης *Player* και ανανεώνουμε την *currentPos* του ανάλογα με την κίνηση αυτή.

Στη συνέχεια, ελέγχουμε αν με αυτή την κίνηση πήρε κάποιο εφόδιο. Αν ναι, τότε δηλώνουμε μια μεταβλητή *sur* ίση με το 1, αλλιώς ίση με το 0. Εάν ο Θησέας πήρε εφόδιο, τότε εξαφανίζουμε το εφόδιο από το ταμπλό.

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  $\text{maxS} < \text{supplieDistance}$  (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  $\text{maxS} > \text{supplieDistance}$  (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.

Στη συνέχεια δεσμεύουμε και αρχικοποιούμε ένα μονοδιάστατο (τύπου *Integer*) πίνακα *bestMove* με τις τιμές των μεταβλητών *finalDice*, *sup*, *supplieDistance* και *minotaurDistance*. Τέλος προσθέτουμε τον πίνακα αυτό στη δομή *path* και επιστρέφουμε την *currentPos*.

- **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 κινήσεων 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 τρέχον 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 τρέχον 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 HeuristicPlayer 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 τρέχον 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 παίζει και οι δύο παίκτες στον τρέχον 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 2η 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, οι κίνηση του κάθε παίκτη, το ταμπλό με τις θέσεις των παικτών (αρχικές και νέες) και των εφοδίων. Τέλος, τυπώνεται το τελικό ταμπλό και τα στατιστικά για τον κάθε γύρο (με την κλήση της συνάρτησης *statistic* της κλάσης *HeuristicPlayer*).