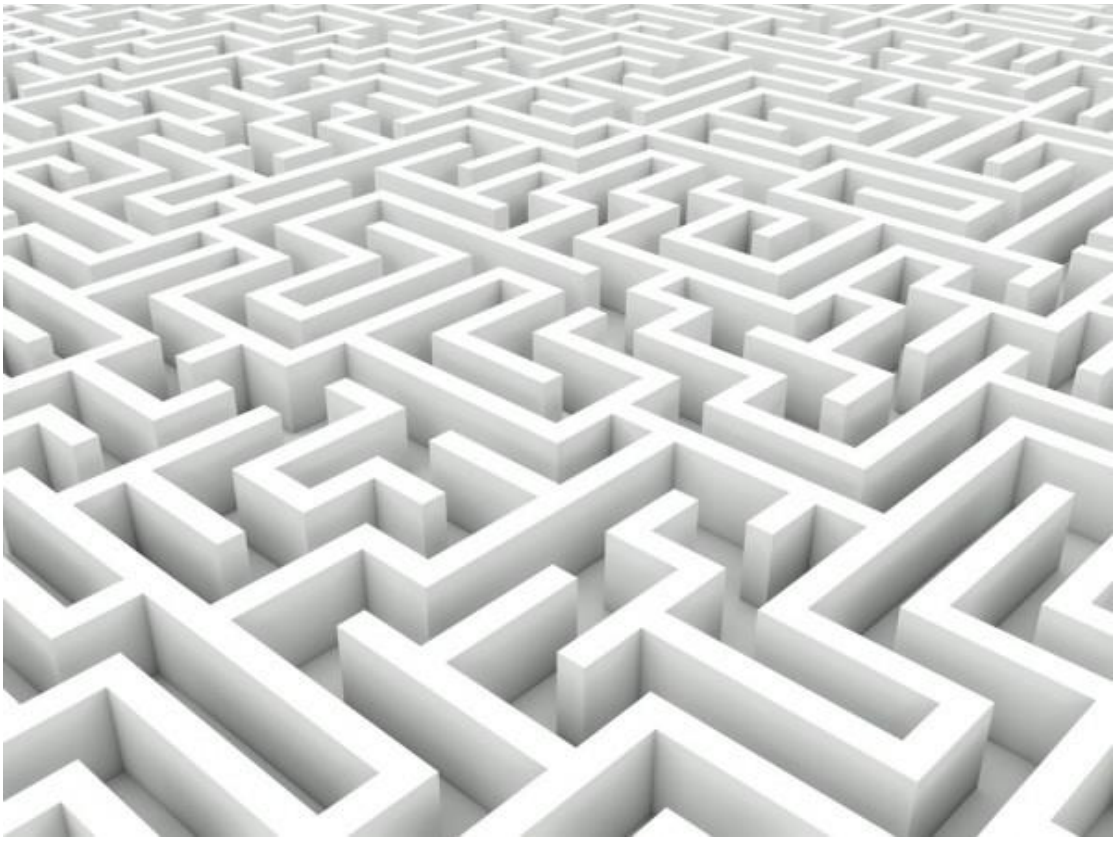
ΕΡΓΑΣΙΑ Γ

Λαβύρινθος, Ο Θησέας και ο Μινώταυρος

Ομάδα 44



Μέλη ομάδας:

Καλαϊτζίδης Ελευθέριος, ΑΕΜ: 10216

Παπαβασιλείου Χαράλαμπος, ΑΕΜ: 9942

Εργασία στα πλαίσια του μαθήματος :

Δομές Δεδομένων- Data Structures

Το πρόβλημα που καλούμαστε να φέρουμε εις πέρας είναι το εξής :

Δημιουργία ενός παιχνιδιού, το οποίο παίζεται πάνω σε ένα ταμπλό, αποτελούμενο από πλακίδια, διαστάσεων «N x N», με N περιττό, το οποίο έχει τοίχους στα πλακίδια, έτσι ώστε να δημιουργείται ένας λαβύρινθος, και εφόδια (λάφυρα) σε κάποια από αυτά. Στο παιχνίδι συμμετέχουν 2 παίχτες (ο Θησέας και ο μινώταυρος).

Στόχος του Θησέα είναι να συγκεντρώσει όλα τα εφόδια στην κατοχή του, ώστε να κερδίσει, ενώ στόχος του μινώταυρου να βρεθεί στο ίδιο πλακίδιο με τον Θησέα, πριν αυτός καταφέρει να συγκεντρώσει όλα τα εφόδια, ώστε να κερδίσει. Εάν δεν γίνει τίποτε από αυτά τα δυο μέχρι το πέρας 100 γύρων τότε το παιχνίδι τελειώνει δίχως νικητή. Κάθε παίκτης μπορεί να κινηθεί οριζόντια ή κατακόρυφα κατά ένα πλακίδιο την φορά, το οποίο καθορίζεται από την ρίψη ζαριού.

Στην οθόνη, το παιχνίδι, θα μας εμφανίζει σε κάθε γύρο την κίνηση των παικτών, το εάν κατάφερε ο Θησέας να πάρει κάποιο εφόδιο και το ταμπλό του παιχνιδιού, με αποτυπωμένα όλα τα στοιχεία (πλακίδια , τοίχους, παίκτες, εφόδια).

Ακολουθεί επεξήγηση του κώδικα.

Κλάση “Supply”:

Σε αυτήν την κλάση ορίζεται το εφόδιο (Supply) με όλα τα χαρακτηριστικά (μεταβλητές) του (supplyId-Id του εφοδίου, x-συνιστώσα x του πλακιδίου στο οποίο βρίσκεται το εφόδιο, y-συνιστώσα y του πλακιδίου στο οποίο βρίσκεται το εφόδιο, supplyTileId – Id του πλακιδίου στο οποίο βρίσκεται το εφόδιο).

Επίσης στην κλάση αυτή υπάρχουν όλες οι απαραίτητες μέθοδοι για την δημιουργία (constructors) ενός αντικειμένου τύπου “Supply”, για την ανάθεση τιμών στις μεταβλητές (setters), και για την ανάγνωση των τιμών που έχουν οι μεταβλητές αυτές (getters).

Κλάση “Tile”:

Σε αυτήν την κλάση ορίζεται το πλακίδιο (Tile) με όλα τα χαρακτηριστικά (μεταβλητές) του (tileId-Id του πλακιδίου, x-συνιστώσα x του πλακιδίου, y-συνιστώσα y του πλακιδίου, up-ύπαρξη ή μη τοίχου στην πάνω πλευρά του πλακιδίου, down-ύπαρξη ή μη τοίχου στην κάτω πλευρά του πλακιδίου, left-ύπαρξη ή μη τοίχου στην αριστερή πλευρά του πλακιδίου, right-ύπαρξη ή μη τοίχου στην δεξιά πλευρά του πλακιδίου).

Επίσης στην κλάση αυτή υπάρχουν όλες οι απαραίτητες μέθοδοι για την δημιουργία (constructors) ενός αντικειμένου τύπου “Tile”, για την ανάθεση τιμών στις μεταβλητές (setters), και για την ανάγνωση των τιμών που έχουν οι μεταβλητές αυτές (getters).

Κλάση “Board”:

Σε αυτήν την κλάση ορίζεται το ταμπλό (Board) με όλα τα χαρακτηριστικά (μεταβλητές) του (N-οριζόντια και κατακόρυφη διάσταση του ταμπλό, S-πλήθος εφοδίων, W-μέγιστος αριθμός τοίχων, tiles-μονοδιάστατος πίνακας διάστασης N*N που περιέχει τα πλακίδια του ταμπλό, supplies-μονοδιάστατος πίνακας που περιέχει τα εφόδια του ταμπλό).

Επίσης στην κλάση αυτή υπάρχουν όλες οι απαραίτητες μέθοδοι για την δημιουργία (constructors) ενός αντικειμένου τύπου “Board”, για την ανάθεση τιμών στις μεταβλητές (setters), και για την ανάγνωση των τιμών που έχουν οι μεταβλητές αυτές (getters).

Πέρα από τις απαραίτητες αυτές μεθόδους υπάρχουν και οι εξής :

- **“createTile”:** σε αυτή την μέθοδο γίνεται η δημιουργία των πλακιδίων πίνακα “tiles” και των τοίχων τους . Αρχικά ορίζουμε να μην έχει τοίχους κανένα πλακίδιο. Στην συνέχεια τοποθετούμε τους εξωτερικούς τοίχους του λαβυρίνθου (γύρω γύρω). Ακολούθως, τοποθετούμε επιπλέον τυχαίους τοίχους στα εξωτερικά πλακίδια του ταμπλό, σύμφωνα με τους περιορισμούς (2 τοίχοι ανά πλακίδιο). Τέλος επιλέγουμε τυχαία εσωτερικά πλακίδια στα οποία τοποθετούμε τυχαία τοίχους σύμφωνα πάντα με τους περιορισμούς. Κάθε φορά που επιλέγουμε τυχαίο πλακίδιο και κάθε φορά που επιλέγουμε τυχαία αν θα προσθέσουμε έναν τοίχο, ελέγχουμε αν αυτό έχει διαθέσιμο περιθώριο για να προσθέσουμε τοίχους με την βοήθεια μιας μεταβλητής “o” (αν έχει τιμή < 2 μπορεί να πάρει τοίχο) και εάν δεν έχουμε ξεπεράσει το όριο των μέγιστων συνολικών τοίχων (W) με την βοήθεια της μεταβλητής “numofw”που μετρά τον αριθμό των τοίχων που έχουμε βάλει μέχρι στιγμής. Κατά την τυχαία επιλογή της πλευράς που θα τοποθετηθεί τοίχος ελέγχουμε εάν το γειτονικό πλακίδιο από την αντίστοιχη πλευρά έχει διαθέσιμο περιθώριο για τοποθέτηση τοίχου (αριθμός τοίχων <2). Εφόσον επιλεχθεί η τοποθέτηση ενός τοίχου σε κάποιο πλακίδιο, θα τοποθετηθεί τοίχος και στην αντίστοιχη θέση του γειτονικού πλακιδίου από την πλευρά που τοποθετήθηκε ο τοίχος (π.χ. έστω ότι πλακίδια 1 και 2 στην ίδια σειρά ,το

1 αριστερά του 2, αν το πλακίδιο 1 έχει τοίχο στα δεξιά του, τότε συνεπάγεται ότι θα έχει τοίχο και το πλακίδιο 2 στα αριστερά του).

- **“createSupply”:** σε αυτή την μέθοδο δημιουργούμε και τοποθετούμε μέσα στον πίνακα “supplies” τα εφόδια κατά αύξοντα αριθμό “id”. Μέσα σε αυτή την μέθοδο δίνονται τυχαία τα x και y τα οποία αντιστοιχούν σε μια θέση του ταμπλό, με τέτοιο τρόπο ώστε να μην αντιστοιχεί η ίδια θέση σε παραπάνω του ενός εφόδια και να μην συμπίπτει η θέση ενός εφοδίου με την αρχική θέση του Θησέα (0) ή του μινώταυρου ((N*N-1)/2-θα μπορούσε και N*N/2 διότι το N είναι περιττός και δίνει το ίδιο αποτέλεσμα-). Ο έλεγχος αυτός γίνεται με την βοήθεια μια μεταβλητής τη “t” η οποία αντιστοιχεί σε τιμή “FALSE”, όταν δεν μπορεί να τοποθετηθεί στην συγκεκριμένη θέση εφόδιο ενώ σε “TRUE” στην αντίθετη περίπτωση.
- **“createBoard”:** η μέθοδος αυτή δημιουργεί τυχαία το ταμπλό του παιχνιδιού χρησιμοποιώντας τις δυο προηγούμενες μεθόδους(“creatTile” και “createSupply”).
- “**getStringRepresentation”:** Αυτή η συνάρτηση επιστρέφει έναν δυσδιάστατο πίνακα διαστάσεων [(2N+1) x 2N] τον οποίο δημιουργούμε “st”. Μέσα σε αυτόν τον πίνακα γίνεται η αποτύπωση του ταμπλό του παιχνιδιού. Κάθε άρτια γραμμή και η μηδενική, αντιστοιχεί σε απεικόνιση των οριζόντιων τοίχων. Κάθε περιττή γραμμή αντιστοιχεί σε απεικόνιση κατακόρυφων τοίχων, παικτών και εφοδίων. Πρώτα τοποθετούμε τους οριζόντιους τοίχους και στην συνέχεια όλα τα αλλά (παίκτες, εφόδια, κατακόρυφοι τοίχοι). Για τους οριζόντιους τοίχους πρώτα γεμίζουμε την μηδενική γραμμή που αντιστοιχεί στον τέρμα κάτω οριζόντιο τοίχο και μετά όλες τις άλλες άρτιες ελέγχοντας αν για την συγκεκριμένη θέση του πίνακα που αντιστοιχεί σε ένα πλακίδιο υπάρχει τοίχος από πάνω. Για να γίνει αυτό διατρέχω τον πίνακα “tails” με την βοήθεια μιας μεταβλητής “k” η οποία αυξάνεται κάθε φορά που αλλάζουμε θέση στον πίνακα “st”, εκτός από την σειρά μηδέν στην οποία χρησιμοποιήθηκε η μεταβλητή “i”της for . Για τις περιττές γραμμές πρώτα ελέγχουμε αν υπάρχει τοίχος ή όχι και στην συνέχεια για κάθε μια περίπτωση ποιος παίκτης υπάρχει στην κάθε θέση ή αν δεν υπάρχει. Για να γίνει αυτό διατρέχω τον πίνακα “tails” με την βοήθεια μιας μεταβλητής “k” η οποία αυξάνεται κάθε φορά που αλλάζουμε θέση στον πίνακα “st”. Όταν δεν υπάρχει παίκτης ή ο παίκτης που υπάρχει είναι ο μινώταυρος ελέγχουμε αν υπάρχει εφόδιο στην θέση αυτή και γίνεται η κατάλληλη απεικόνιση. Επίσης ελέγχουμε αν βρισκόμαστε σε θέση που αντιστοιχεί σε πλακίδιο τέρμα δεξιά ώστε να τοποθετήσουμε τον εξωτερικό τοίχο (σε όλες τις γραμμές).

Κλάση "Player":

Σε αυτήν την κλάση ορίζεται ο παίκτης (Player) με όλα τα χαρακτηριστικά (μεταβλητές) του (playerId-το Id του παίκτη, name-όνομα παίκτη, board-ταμπλο του παιχνιδιου, score- το σκορ του παίκτη, x-συνιστώσα x του παίκτη στο ταμπλό ,y-συνιστώσα y του παίκτη στο ταμπλό). Επίσης στην κλάση αυτή υπάρχουν όλες οι απαραίτητες μέθοδοι για την δημιουργία (constructors) ενός αντικειμένου τύπου "Player", για την ανάθεση τιμών στις μεταβλητές (setters), και για την ανάγνωση των τιμών που έχουν οι μεταβλητές αυτές (getters).

Πέρα από τις απαραίτητες αυτές μεθόδους υπάρχουν και η εξής :

- **"move":** Στην μέθοδο αυτή οι παίκτες «ρίχνουν ζάρι» και επιλέγουν τυχαία μια από τις διαθέσιμες κινήσεις (εμπρός, πίσω, δεξιά, αριστερά). Αρχικά δημιουργούμε έναν πίνακα "m" τεσσάρων θέσεων και έναν πίνακα "p" ίδιο με τον "tiles" του "board". Για την επιλογή της τυχαίας κίνησης μας βοηθά μια μεταβλητή "a" η τιμή της οποίας καθορίζει την κατεύθυνση (1-επάνω, 3-δεξιά, 5-κάτω, 7-αριστερά). Στην συνέχεια ελέγχει αν μπορεί να γίνει η κίνηση (με χρήση if). Για να γίνει αυτός ο έλεγχος, χρησιμοποιούμε τις μεταβλητές "u", "d", "l", "r" οι οποίες δείχνουν το αν υπάρχει τοίχος πάνω κάτω αριστερά και δεξιά αντίστοιχα (αν υπάρχει τοίχος παίρνουν την τιμή "TRUE" και αν δεν υπάρχει "FALSE"). Ανάλογα τον παίκτη, την τιμή του "a" και του "u" ή "d" ή "l" ή "r", αν μπορεί να παιχτεί, η κίνηση θα αλλάξει το "x" και το "y" του αντίστοιχου παίκτη κατάλληλα, αλλιώς θα εμφανίσει μήνυμα το οποίο μας λέει ότι δεν μπορεί να κινηθεί. Ακόμη αν ο παίκτης είναι ο Θησέας και μπορεί να κινηθεί ελέγχει αν (if) υπάρχει εφόδιο στο πλακίδιο στο οποίο κινείται (διατρέχοντας τον πίνακα "supplies"του "board"με την βοήθεια μιας for) και αν υπάρχει μας εμφανίζει μήνυμα πως ο Θησέας πήρε ένα εφόδιο και τοποθετεί το "Id" του εφοδίου στο "m[3]". Τέλος μας επιστρέφει τον πίνακα "m" στις θέσεις(m[0], m[1], m[2]) του οποίου πρώτα έχουν τοποθετηθεί το "Id", το "x", το "y", αντίστοιχα, του πλακιδίου στο οποίο πάει ο παίκτης.

Κλάση "HeuristicPlayer":

Σε αυτήν την κλάση ορίζεται ο έξυπνος παίκτης "HeuristicPlayer ". Η κλάση αυτή κληρονομεί την κλάση "Player", οπότε έχει όλες τις μεθόδους και τις μεταβλητές της. Η κλάση αυτή περιλαμβάνει επιπλέον μια λίστα "arraylist" με όνομα "path" αυτήν στην οποία αποθηκεύονται διάφορα στοιχεία για την κίνηση του παίκτη. Στην κλάση περιλαμβάνονται όλες οι απαραίτητες μέθοδοι για την δημιουργία "constructors", για

την κατασκευή ενός αντικειμένου τύπου “HeuristicPlayer ” (μια κενή, χωρίς ορίσματα, και μια με όλα τα απαραίτητα ορίσματα της κλάσης “player”).

Πέρα από τις απαραίτητες αυτές μεθόδους υπάρχουν και η εξής :

- **“evaluate”:** Στην μέθοδο αυτή γίνεται η αξιολόγηση κάποιας πιθανής κίνησης του παίκτη. Η μέθοδος αυτή δέχεται ως ορίσματα την θέση του παίκτη, την ζαριά και την θέση του αντιπάλου (currentPos, dice, minotaurtileid).

Διακρίνουμε 5 περιπτώσεις με ελέγχους (if), για την ζαριά (1,3,5,7,0) και μια ειδική περίπτωση (αν το τελευταίο εφόδιο βρίσκεται στο ίδιο πλακίδιο με τον αντίπαλο δίπλα από τον παίκτη μας). Αν το ζάρι έχει την τιμή 1, τότε ελέγχουμε προς την επάνω κατεύθυνση ένα προς ένα τα επόμενα 3 πλακίδια (αν έχουν τοίχο προς την κατεύθυνση αυτή, και αν δεν έχουν, αν περιλαμβάνουν εφόδιο ή αντίπαλο). Το ίδιο κάνει και στις περιπτώσεις που το ζάρι είναι 3,5,7 προς τα δεξιά κάτω και αριστερά αντίστοιχα. Αν προς την κατεύθυνση που ελέγχουμε υπάρχει εφόδιο αυξάνουμε την μεταβλητή που αντιστοιχεί στα εφόδια (s) κατά 1 αν αυτό βρίσκεται στο διπλανό πλακίδιο, 0,5 αν υπάρχει ένα πλακίδιο ανάμεσα, 0,3 αν υπάρχουν δυο πλακίδια ανάμεσα. Αν προς την κατεύθυνση που ελέγχουμε υπάρχει εχθρός θέτουμε στη μεταβλητή που αντιστοιχεί στον εχθρό(m) την τιμή -1 αν αυτός βρίσκεται στο διπλανό πλακίδιο, -0,5 αν υπάρχει ένα πλακίδιο ανάμεσα , -0,3 αν υπάρχουν δυο πλακίδια ανάμεσα. Στην συνέχεια χρησιμοποιούμε μια συνάρτηση (s*0.46 + m*0.54) για να βγάλουμε τους πόντους της κίνησης, τους οποίους αποθηκεύουμε σε μια μεταβλητή (points). Στην περίπτωση που έχει πάρει όλα τα εφόδια εκτός από ένα, αν το εφόδιο βρίσκεται στο ίδιο πλακίδιο με τον αντίπαλο το οποίο είναι δίπλα από τον παίκτη τότε αυτό θέτει του πόντους της κίνησης 100 (points = 100). Στην περίπτωση που το ζάρι έχει τιμή 0 θέτει τους πόντους της κίνησης -100 (points = -100). Τέλος επιστρέφουμε τους πόντους της κίνησης (return points).

- **“getNextMove”:** Στην μέθοδο αυτή γίνεται η επιλογή της καλύτερης κίνησης. Σαν ορίσματα παίρνει την θέση του παίκτη και την θέση του αντιπάλου Στην αρχή δημιουργούμε έναν πίνακα (eval) 4 θέσεων, έναν (κ, τύπου Integer) και έναν (p) 8 θέσεων τις πρώτες τέσσερις(0, 1, 2, 3) βάζουμε τις πιθανές ζαριές 1, 3, 5, 7 αντίστοιχα. Ελέγχω προς όλες τις κατευθύνσεις εάν υπάρχει τοίχος, και αν υπάρχει θέτω το ζάρι μηδέν (αλλάζοντας την τιμή στην αντίστοιχη θέση του πίνακα). Στην συνέχεια τοποθετώ στις επόμενες θέσεις του πίνακα την αξιολόγηση για κάθε πιθανή κίνηση (στις θέσεις 4, 5, 6, 7, για την κίνηση προς τα πάνω, δεξιά, κάτω, αριστερά

αντίστοιχα). Αντιγράφουμε τις αξιολογήσεις στον πίνακα “eval”. Τοποθετώ τα στοιχεία του πίνακα “eval” σε φθίνουσα σειρά (με την βοήθεια δυο for). Ελέγχω αν η μεγαλύτερη τιμή είναι μοναδική, και αν είναι βρίσκω σε ποια θέση του πίνακα “p” αντιστοιχεί. Έτσι βρίσκω και την κίνηση της οποίας η αξιολόγηση είναι μεγαλύτερη και μετακινούμε τον παίκτη κατάλληλα. Τοποθετούμε στην $1^{η}$ θέση του πίνακα “k”(k[0]) την τιμή του ζαριού που επιλέξαμε. Επίσης ελέγχουμε αν αυξήθηκε το σκορ του παίκτη (αν πήρε δηλαδή εφόδιο), και αν αυξήθηκε τότε τοποθετώ στην $2^{η}$ θέση του πίνακα “k” (κ[1])την τιμή 1 αλλιώς την τιμή 2 (με την βοήθεια if και else). Ακόμα ελέγχουμε εάν στην νέα θέση υπάρχει εφόδιο και αν υπάρχει αυξάνουμε το σκορ και “παίρνουμε” το εφόδιο (το τοποθετούμε εκτός του ταμπλό). Μετά ελέγχουμε αν υπάρχει γύρω από την νέα θέση εχθρός (αν δεν υπάρχουν τοίχοι ώστε να μπορεί ο παίκτης να βλέπει) και αν υπάρχουν εφόδια (αν δεν υπάρχουν τοίχοι ώστε να μπορεί ο παίκτης να βλέπει). Αν υπάρχει εχθρός αποθηκεύουμε την απόσταση του από τον παίκτη σε μια μεταβλητή (n, αρχικοποιημένη στο -1) η οποία τοποθετείται στην $4^{η}$ θεση του πίνκα “k”(k[3]), και αν υπάρχουν/ει εφόδια/εφόδιο αποθηκεύουμε την απόσταση του πιο κοντινού από τον παίκτη σε μια μεταβλητή (l, αρχικοποιημένη στο 10). Αν η τιμή το “l” δεν έχει αλλάξει (είναι 10) τότε το θέτουμε -1. Τοποθετούμε την τιμή του “l” στην $3^{η}$ θέση του πίνακα “k” (k[2]).τοποθετούμε σε μια κενή θέση του “path” τον πίνακα “k”.και επιστρέφουμε το “id” της νέας θέσης (m) του παίκτη.

- **Stat:** Στην μέθοδο αυτή τυπώνουμε για τον τελευταίο γύρο “a”(a=path.size()) (μέχρι την στιγμή που καλείται) το ζάρι, ο γύρος, το σκορ και οι αποστάσεις από το κοντινότερο εφόδιο και εχθρό (αν υπάρχουν στο οπτικό του πεδίο). Όλα αυτά τα στοιχεία τα αντλούμε από την “path” (με τις συναρτήσεις path.get(a-1)[1], path.get(a-1)[2] , path.get(a-1)[3]) και από το “score” του παίκτη.
- **Statistics:** Στην μέθοδο αυτή(η οποία καλείται στο τέλος του παιχνιδιού), θέτουμε στην μεταβλητή “a” τον αριθμό των γύρων που έπαιξε ο παίκτης (a=path.size()) τυπώνουμε για κάθε γύρο “i”(για i από 0 εως a -for-) το ζάρι, ο γύρος, το σκορ και οι αποστάσεις από το κοντινότερο εφόδιο και εχθρό (αν υπάρχουν στο οπτικό του πεδίο). Όλα αυτά τα στοιχεία τα αντλούμε από την “path” (με τις συναρτήσεις path.get(a-1)[1], path.get(a-1)[2] , path.get(a-1)[3]) και από το “score” του παίκτη και αθροίζουμε τις φορές που ο παίκτης κινήθηκε πάνω (με την βοήθεια της μεταβλητής θ), κάτω (με την βοήθεια της μεταβλητής d), δεξιά (με την βοήθεια

της μεταβλητής r), αριστερά (με την βοήθεια της μεταβλητής l) και τυπώνουμε τα αθροίσματα αυτά καθώς και το σκορ του παίκτη (score).

Κλάση "Node":

Σε αυτήν την κλάση ορίζεται ο κόμβος (Node) με όλα τα χαρακτηριστικά (μεταβλητές) του (parent(κόμβος-πατέρας του νέου κόμβου), children (λίστα που περιέχει τους κόμβους-παιδιά του νέου κόμβου), nodeDepth(το βάθος του κόμβου), nodeMove(πίνακας που περιέχει τις νέες συντεταγμένες και την ζαριά της κίνησης), nodeBoard (το ταμπλό του παιχνιδιού), nodeEvaluation(η αξιολόγηση της κίνησης)).

Επίσης στην κλάση αυτή υπάρχουν όλες οι απαραίτητες μέθοδοι για την δημιουργία (constructors) ενός αντικειμένου τύπου " Node ", για την ανάθεση τιμών στις μεταβλητές (setters), και για την ανάγνωση των τιμών που έχουν οι μεταβλητές αυτές (getters).

Κλάση " MinMaxPlayer ":

Σε αυτήν την κλάση ορίζεται ένας νέος παίκτης (MinMaxPlayer) ο οποίος κληρονομεί με όλα τα χαρακτηριστικά (μεταβλητές) του player και επιπλέων έχει μια λίστα "path" στην οποία αποθηκεύονται πληροφορίες για τις κινήσεις του. Επίσης στην κλάση αυτή υπάρχουν όλες οι απαραίτητες μέθοδοι για την δημιουργία (constructors) ενός αντικειμένου τύπου " MinMaxPlayer ", για την ανάθεση τιμών στις μεταβλητές (setters), και για την ανάγνωση των τιμών που έχουν οι μεταβλητές αυτές (getters).

Πέρα από τις απαραίτητες αυτές μεθόδους υπάρχουν και η εξής :

- **"evaluate":** Στην μέθοδο αυτή γίνεται η αξιολόγηση μίας από τις πιθανές κινήσεις .ο τρόπος αξιολόγησης είναι ο ίδιος με αυτόν στην κλάση " HeuristicPlayer "
- **"getNextMove":** Στην μέθοδο αυτή γίνεται η επιλογή της καλύτερης κίνησης με την βοήθεια της δημιουργίας ενός δέντρου. Αρχικά δημιουργούμε την ρίζα

“root”του δέντρου και στην συνέχεια το υπόλοιπο δέντρο (το οποίο είναι ένα min-max tree) με την βοήθεια της “createMySubtree”. Επιλέγουμε την καλύτερη κίνηση αξιοποιώντας την συνάρτηση “chooseMinMaxMove”.καλούμε την συνάρτηση moven έτσι ώστε να δημιουργήσουμε έναν πίνκα με πληροφορέιες για την νέα θεση (η moven είναι η συνάρτηση move της ενδεικτικής απάντησης της πρώτης εργασίας λίγο τροποποιημένη και προστέθηκε στην υπάρχουσα κλάση “Player” όπως επίσης και η συνάρτηση “whereTo”). Στην συνέχια φτιάχνουμε έναν πίνακα 4 θέσεων και στην πρώτη του θέση τοποθετούμε την ζαριά (στις άλλες μπαίνουν πληροφορίες για την απόσταση από εφόδιο, αντίπαλο και πόσα εφόδια έχει μαζέψει).με τον ίδιο τρόπο όπως στην κλάση “ HeuristicPlayer ”ελέγχουμε τις αποστάσεις από εφόδια, αντίπαλο και πόσα εφόδια έχει μαζέψει τα τοποθετούμε στον πίνακα και στην συνέχεια τοποθετούμε τον πίνακα μέσα στην λίστα μας.

- **“createMySubtree”:** Στην μέθοδο αυτή γίνεται η δημιουργία του μέρους του δέντρου που βρίσκεται κάτω από την ρίζα. Ελέγχουμε για κάθε κατεύθυνση αν υπάρχει τοίχος (if). Αν δεν υπάρχει τότε αυξάνουμε μία μεταβλητή που αντιστοιχεί στον αριθμό των πιθανών κινήσεων “numofmoves”. Φτιάχνουμε έναν πίνακα τριών θέσεων που περιέχει την τετμημένη την τεταγμένη και τον αριθμό της ζαριάς, δημιουργούμε ένα αντίγραφο του πίνακα, μία νέα κενή λίστα (η οποία θα χρησιμοποιηθεί σαν λίστα του νέου κόμβου) και δημιουργούμε τον νέο κόμβο. Στην συνέχια προσθέτουθμε τον κόμβο που δημιουργήσαμε σαν παιδί της ρίζας και δημιουργούμε το μέρος του δέντρου που αντιστοιχεί στο επόμενο βάθος(δεύτερο) . Η διαδικασία επαναλαμβάνεται για όλες τις πιθαμές κινήσεις.

- **“createOpponentSubtree”:** Στην μέθοδο αυτή γίνεται η δημιουργία του μέρους του δέντρου που αντιστοιχεί σε βάθος 2. Ο τρόπος δημιουργίας του είναι αντίστοιχος με την δημιουργία του μέρους του δέντρου που αντιστοιχεί σε βάθος 1 και επαναλαμβάνεται για κάθε πιθανή κίνηση του αντιπάλου(η τελική αξιολόγηση κάθε φύλλου αντιστοιχεί με το άθροισμα της αξιολόγησης του κόμβου πατέρα και της κίνησης).
- **“chooseMinMaxMove”:** Στην μέθοδο αυτή γίνεται η δημιουργία του αλγορίθμου που μας βοηθά να κατασκευάσουμε το “min-max tree” και έτσι να πάρουμε την σωστή κίνηση. Με την βοήθεια μίας δομής επανάληψης (for) ελέγχουμε για κάθε κόμβο που αντιστοιχεί στο πρώτο επίπεδο του δέντρου ποιος από τους κόμβους-παιδιά του έχει μικρότερη τιμή αξιολόγησης (με την βοήθεια δεύτερης

δομής επανάληψης “for”) έτσι ώστε να επιλεχθεί σαν ελάχιστο και να αντικαταστήσει την τιμή της αξιολόγησης του πατέρα του. Με αντίστοιχο τρόπο γίνεται και η επιλογή της μεγαλύτερης αξιολόγησης μεταξύ των παιδιών της ρίζας ώστε να επιστραφεί η σωστή κίνηση (αυτή με την μεγαλύτερη αξιολόγηση).

Οι συναρτήσεις “stat()” και “ statistics()” είναι ίδια με τις αντίστοιχες της κλάσης “HeuristicPlayer”.

Κλάση “Game”:

Σε αυτήν την κλάση ορίζεται το παιχνίδι (Game) με το χαρακτηριστικό (μεταβλητή) του (round-γύρος του παιχνιδιού). Επίσης στην κλάση αυτή υπάρχουν όλες οι απαραίτητες μέθοδοι για την δημιουργία (constructor), ενός αντικειμένου τύπου “Game ”, για την ανάθεση τιμής στην μεταβλητή (setter), και για την ανάγνωση της τιμής που έχει η μεταβλητή αυτή (getter).

Επίσης η κλάση αυτή περιλαμβάνει την “main”. Σε αυτή αρχικά δημιουργούμε τρείς μεταβλητές “w”, “n”, “s”, οι οποίες παίρνουν τις τιμές “0”, διάσταση του ταμπλό και αριθμό εφοδίων αντίστοιχα. Στην συνέχεια δημιουργούμε ένα αντικείμενο “g” τύπου “Game”, ένα “p” τύπου “Board”(ταμπλό) με ορίσματα του “constructor” «n(διάσταση του ταμπλό), s (αριθμό εφοδίων), (n * n * 3 + 1) / 2 (μέγιστος αριθμός τοίχων)», και γεμίζουμε το ταμπλό τυχαία με την βοήθεια της “createBoard”(p.creatBoard();). Ακόμα δημιουργούμε ένα αντικείμενο τύπου “Player”, “player2” με ορίσματα του “constructor” «2(id παίκτη), ”Minotaur” (όνομα παίκτη), p(το ταμπλό που δημιουργήσαμε προηγουμένως), 0(σκορ του μινώταυρου), 0(x του μινώταυρου) , 0(y του μινώταυρου)» και ένα τύπου “MinMaxPlayer”, “player1” με ορίσματα του “constructor” «1(id παίκτη),”Theseus” (όνομα παίκτη), p(το ταμπλό που δημιουργήσαμε προηγουμένως), 0(σκορ του Θησέα), 0(x του Θησέα) , 0(y του Θησέα)» και τυπώνουμε το ταμπλό στην αρχική του κατάσταση.

Η τύπωση αυτή γίνεται με την βοήθεια δυο δομών επανάληψης “for” (for (int k=2*n;k>-1;k--) και for(int j=0;j<n;j++) η δεύτερη μέσα στην πρώτη) και τυπώνοντάς κάθε θέση του πίνακα που αποτελεί αναπαράσταση του ταμπλό με την εντολή “System.out.print (p.getStringRepresentation (player1.getX()*n + player1.getY(), player2.getX()*n + player2.getY())[k][j]) ”στην εσωτερική for και αλλαγή σειράς στην εξωτερική. Με αυτόν τον τρόπο κατορθώνομε να εκτυπώσουμε τις θέσεις του πίνακα από κάτω προς τα πάνω και από αριστερά προς τα δεξιά και να έχουμε ορθή απεικόνιση του ταμπλό.

Ακολούθως κλείνουμε την είσοδο (από εκεί που μπήκε ο Θησέας στον λαβύρινθο) με την εντολή " p.getTiles()[0].setDown(true); ". Βάζουμε μια δομή επανάληψης "for", ώστε το παιχνίδι ναι παίζεται για 100 γύρους, εκτός αν σταματήσει από άλλο παράγοντα. Μέσα σε αυτή τη "for" τυπώνουμε τον γύρο στο οποίο βρισκόμαστε κάθε φορά , αντιγράφουμε τον "player1" και τον "player2" στις νέες μεταβλητές "p1" και "p2" αντίστοιχα, καλούμε την "getNextMove " για τον "player1" και ελέγχουμε προς τα ποια μεριά κινήθηκε κάθε φορά, ώστε να τυπώσουμε σωστά την κίνηση που έκανε και καλούμε την "stat" ώστε να τυπωθούν περισσότερα στοιχεία για την κίνηση του Θησέα.

Ακολούθως ελέγχουμε με "if" εάν έχει κατορθώσει να συγκεντρώσει όλα τα εφόδια ώστε να τελειώσει το παιχνίδι και να κερδίσει και με μια άλλη "if" εάν κινήθηκε σε ίδιο πλακίδιο με τον "player2"(μινώταυρο), οπότε να τελειώσει το παιχνίδι με νικητή τον "player2". Με τον ίδιο τρόπο παίζει και ο "player2"(αν δεν έχει τελειώσει ήδη το παιχνίδι). Τυπώνεται η κίνηση του και ελέγχουμε με μια "if" αν κινήθηκε σε πλακίδιο που βρίσκεται ο "player1" ώστε να τελειώσει το παιχνίδι με νικητή τον "player2". Εφόσον δεν έχουμε κάποιον νικητή ακόμα τυπώνουμε το ταμπλό με τον τρόπο που αναφέραμε προηγουμένως.

Έξω από την "for" ελέγχουμε εάν νίκησε ο Θησέας (if(w==1)), ώστε να τυπώσουμε το ταμπλό στην τελική του μορφή μόνο με το Θησέα απεικονισμένο, πληροφορίες για τις κινήσεις του Θησέα σε όλο το παιχνίδι καθώς και στατιστικά στοιχεία για τις κινήσεις αυτές καλώντας την "statistics" και το μήνυμα ότι νίκησε. Αντίστοιχα ελέγχουμε αν νίκησε ο μινώταυρος (else if(w==2)), ώστε να τυπώσουμε το ταμπλό στην τελική του μορφή μόνο με το μινώταυρο απεικονισμένο, πληροφορίες για τις κινήσεις του Θησέα σε όλο το παιχνίδι καθώς και στατιστικά στοιχεία για τις κινήσεις αυτές καλώντας την "statistics" και το μήνυμα ότι νίκησε, αλλιώς (else) τυπώνουμε πληροφορίες για τις κινήσεις του Θησέα σε όλο το παιχνίδι καθώς και στατιστικά στοιχεία για τις κινήσεις αυτές καλώντας την "statistics" και το μήνυμα πως δεν κέρδισε κανείς.

Σημείωση : όλη η εργασία έγινε σύμφωνα με τις συντεταγμένες (x,y) της μορφής που μας δόθηκαν(κατακόρυφος άξονας = άξονας των x, οριζόντιος άξονας = άξονας των y). Στο τέλος, στην εκτύπωση, οι θέσεις των παικτών έγινε σύμφωνα με την συνήθη από τα μαθηματικά μορφή (x,y) (οριζόντιος άξονας = άξονας των x, κατακόρυφος άξονας = άξονας των y). Στην κλάση "HeuristicPlayer" έχουμε δημιουργήσει μία

επιπλέον συνάρτηση από αυτές που ζητούνται στην εκφώνηση, την “Stat”. Στην συνάρτηση “getNextMove” δεν έγινε χρήση της συνάρτησης “move” του “Player”, και στην “evaluate” δεν μετακινούμε τον παίκτη αλλά ελέγχουμε απλά τις θέσεις του ταμπλό. Αυτά που αναφέρονται σχετικά με την κλάση “HeuristicPlayer” ήταν διευκρινίσεις που δόθηκαν αφού είχαμε γράψει το μεγαλύτερο μέρος του κώδικα και μας είπατε να επιλέξουμε αυτό τον τρόπο υλοποίησης στις απορίες που τέθηκαν στις διαλέξεις .Στην εργασία Γ δεν χρησιμοποιείται η κλάση “HeuristicPlayer”.

Στοιχεία επικοινωνίας :

Καλαϊτζίδης Ελευθέριος:

ΤΗΛ: 6981494232

email: eleftherk@ece.auth.gr

Παπαβασιλείου Χαράλαμπος :

ΤΗΛ: 6984219213

email: papavasic@ece.auth.gr