**ΕΡΓΑΣΙΑ 2 ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ.**

**ΗΛΕΚΤΡΟΝΙΚΗ ΑΝΑΦΟΡΑ.**

**Κελέση Ελπίδα**

**ΑΕΜ:9410**

**Καλαμπούκα Ευαγγελία**

**ΑΕΜ:9411**

**Πρόβλημα:**

Σε αυτό το μέρος της εργασίας μας ανατέθηκε να δημιουργήσουμε μία καινούρια κλάση για τον παίκτη που παίζει με στρατηγική.

**Οι κλάσεις:**

**α)Κλάση “HeuristicPlayer”**

Η κλάση αυτή κληρονομεί την κλάση “Player” και περιλαμβάνει την παρακάτω μεταβλητή:

* “path ”, “arraylist” με στοιχεία πίνακες ακεραίων, όπου σε κάθε πίνακα αποθηκεύονται η ζαριά του παίκτη, οι πόντοι του, τα βήματα του, οι σκάλες που ανέβηκε και τα φίδια που τον έφαγαν .

Και τις εξής δημόσιες μεθόδους:

* Τις δύο συναρτήσεις αρχικών συνθηκών(μία χωρίς ορίσματα, μία με ορίσματα).
* Μία συνάρτηση “evaluate(int currentPos, int dice)”, η οποία έχει ως ορίσματα τη θέση του παίκτη και τον αριθμό του ζαριού και επιστρέφει μία τιμή τύπου “double”, η οποία αντιστοιχεί στην συνάρτηση αξιολόγησης για τη συγκεκριμένη κίνηση.
* Αρχικά η μεταβλητή “finalPos” παίρνει την τιμή “(currentPos + dice)”.
* Στη συνέχεια ακολουθεί ένας βρόχος “do-while”, ο οποίος επαναλαμβάνεται όσο η μεταβλητή “check” έχει την τιμή “true”.
* Δίνεται η τιμή “false” στη μεταβλητή “check”.
* Ξεκινά μία επανάληψη “for”, η οποία τρέχει τόσες φορές όσο είναι και το μέγεθος του πίνακα “apples” της κλάσης “Board”:
* Σε μία “if”, ελέγχεται αν στη θέση του παίκτη υπάρχει μήλο. Αν υπάρχει αυξάνεται η μεταβλητή “po”, κατά τους πόντους του μήλου.
* Ξεκινά μία επανάληψη “for”, η οποία τρέχει τόσες φορές όσο είναι και το μέγεθος του πίνακα “snakes” της κλάσης “Board”:
* Σε μία “if”, ελέγχεται αν στη θέση του παίκτη υπάρχει φίδι. Αν υπάρχει, δίνεται η τιμή “true” στη μεταβλητή “check”, και η τελική θέση του παίκτη γίνεται ίση με τη θέση της ουράς του αντίστοιχου φιδιού.
* Με μία “if”, εξετάζεται η τιμή της μεταβλητής “check”.
* Αν αυτή είναι “true”, τότε με την εντολή “continue” το πρόγραμμα συνεχίζει να τρέχει από την αρχή της “do-while” και ελέγχει ξανά για την ύπαρξη μήλου και φιδιού στη νέα θέση του παίκτη.
* Αλλιώς συνεχίζει με μία επανάληψη “for””, η οποία τρέχει τόσες φορές όσο είναι και το μέγεθος του πίνακα “ladders” της κλάσης “Board”:
* Με μία “if”, ελέγχεται αν υπάρχει κάτω μέρος σκάλας στη θέση του παίκτη. Αν υπάρχει, δίνεται η τιμή “true” στη μεταβλητή “check”, έτσι ώστε να συνεχίσει να τρέχει η “do-while”, η θέση του παίκτη γίνεται ίδια με τη θέση από το πάνω μέρος της σκάλας.
* Με βάση την τελική θέση του παίκτη και τους πόντους του βρίσκεται μέσω της συνάρτησης στόχου, η αξιολόγηση της συγκεκριμένης κίνησης.
* Επιστρέφεται η τιμή της αξιολόγησης.
* Μια συνάρτηση “getNextMove(int currentPos)”, η οποία έχει ως όρισμα τη θέση του παίκτη και τύπο επιστροφής “int”.
  + Αρχικά με μία “for” που τρέχει για τις έξι διαφορετικές ζαριές, αποθηκεύεται στην αντίστοιχη θέση του πίνακα “moves”, η επιστρεφόμενη αξιολόγηση της συνάρτησης “evaluate” για τη συγκεκριμένη ζαριά. (π.χ. στη θέση 4 του πίνακα αποθηκεύεται η αξιολόγηση όταν το ζάρι είναι 4)
  + Με έναν βρόχο “for”, που τρέχει για τις έξι διαφορετικές ζαριές και μία εμφωλευμένη “if”, εξετάζεται ποια ζαριά έχει την καλύτερη αξιολόγηση και αυτή αποθηκεύεται στην μεταβλητή “finaldie”.
  + Εκχωρείται η τιμή του ζαριού (“finaldie”), στη θέση 0 του πίνακα Round.
  + Στον πίνακα “move”, αποθηκεύεται ο επιστρεφόμενος πίνακας της συνάρτησης “move” για την επιλεγμένη ζαριά.
  + Στη μεταβλητή “evaluation”, εκχωρείται η αξιολόγηση της τελικής ζαριάς μέσω της συνάρτησης “evaluate”. Στη θέση 1 του πίνακα Round αποθηκεύονται οι πόντοι του παίκτη με την επιλεγμένη ζαριά.
  + Στη θέση 2 του πίνακα Round αποθηκεύονται με τη βοήθεια των στοιχείων του πίνακα “move”, τα βήματα του παίκτη (τελική θέση – αρχική θέση).
  + Στη θέση 3 του πίνακα Round αποθηκεύονται, με τη βοήθεια των στοιχείων του πίνακα “move” τα μήλα που έφαγε ο παίκτης (κόκκινα και μαύρα μήλα).
  + Στη θέση 4 του πίνακα Round αποθηκεύονται, με τη βοήθεια των στοιχείων του πίνακα “move”, τα φίδια που έφαγαν τον παίκτη.
  + Στη θέση 5 του πίνακα Round αποθηκεύονται, με τη βοήθεια των στοιχείων του πίνακα “move”, οι σκάλες που ανέβηκε ο παίκτης.
  + Στη θέση 6 του πίνακα αποθηκεύονται τα κόκκινα μήλα που έφαγε ο παίκτης.
  + Προστίθεται ο πίνακας “Round”, στη μεταβλητή “path”.
  + Επιστρέφεται μέσω του πίνακα “move”, η τελική θέση του παίκτη.
* Μία συνάρτηση “statistics”, χωρίς ορίσματα και με τύπο επιστροφής “void”.
  + Αρχικά εκτυπώνονται με τη βοήθεια της μεταβλητής “path” στοιχεία για το συγκεκριμένο γύρο. Ο αριθμός του γύρου ισούται με το μέγεθος της “arraylist” “path”, αφού σε κάθε γύρο προστίθεται και ένα στοιχείο σε αυτή.
  + Με μία “for” που τρέχει τόσες φορές όσοι είναι και οι γύροι του παιχνιδιού μέχρι εκείνη τη στιγμή, αποθηκεύονται στις αντίστοιχες μεταβλητές στατιστικά στοιχεία του γύρου, δηλαδή επισκέψεις σε πλακίδιο με κεφάλι φιδιού, βάση σκάλας, κόκκινο ή μαύρο μήλο.
  + Τέλος εκτυπώνονται αυτά τα στατιστικά στοιχεία.

**ε)Κλάση “Game”**

Η κλάση αυτή περιλαμβάνει τις εξής μεταβλητές:

* “round”, τύπου “int”

Επιπλέον περιλαμβάνει και τις παρακάτω μεθόδους:

* Τις τρεις συναρτήσεις αρχικών συνθηκών (μία χωρίς ορίσματα και μια με όρισμα τη μεταβλητή round, τύπου int).
* Όλες τις “set” και “get” για τις μεταβλητές που αναφέρθηκαν.
* Ορίζεται ένα hashmap με όνομα first, με ορίσματα integer και double, στο οποίο πρόκειται να μπουν οι τιμές των id των παικτών σαν κλειδιά και σαν values οι ζαριές που έφεραν με ακέραιο ψηφίο την πρώτη ζαριά, πρώτο δεκαδικό τη δεύτερη κλπ(η κάθε μία από τις οποίες θα υπάρχουν σε περίπτωση που φέρει ίδια ζαριά ή ζαριές κάποιος παίκτης με κάποιον ή κάποιους άλλους).
* Ορίζεται και αρχικοποιείται μία μεταβλητή global με όνομα min\_iter, η οποία δείχνει τις φορές που έριξε ο παίκτης ξανά το ζάρι, ώστε να καθοριστεί η σειρά που θα παίξει.
* Μία συνάρτηση Map<Integer,Integer> setTurns() με όρισμα το ArrayList<Object> players, η οποία έχει μέσα της δύο συναρτήσεις: την void checkRow() , που λειτουργεί αναδρομικά, με ορίσματα int row, int iter, int[][] binaryTable και την ArrayList<Integer> checkTable() με όρισμα int[][] binaryTable. Η setTurns() επιστρέφει ένα ταξινομημένο map με τα id των παικτών και την ή τις πρώτες ζαριές τους(πολλές σε περίπτωση που ρίξουν το ίδιο με κάποιον ή κάποιους άλλους παίκτες ζάρι), ώστε να ξέρουμε τη σειρά με την οποία θα παίξουν. Δηλαδή, ρίχνουν όλοι οι παίκτες μια φορά το ζάρι και ταξινομούνται. Αν κάποιος ρίξει ίδιο ζάρι με κάποιον ή κάποιους άλλους, αυτοί ξαναρίχνουν το ζάρι και ούτω κάθε εξής, μέχρι να μην έχουμε πολλαπλές ίδιες ζαριές στον ίδιο γύρο. Έτσι θα διευκρινιστεί η σειρά με την οποία θα παίξουν, χωρίς να αλλάζει η αρχική διάταξη αυτών που είχαν εξ ’αρχής ή στη συνέχεια διαφορετική-μοναδική ζαριά. Για παράδειγμα:   
  αν αρχικά έχουμε πχ. 6 παίχτες και ρίξουν τις ζαριές (σειρά με βάση τα id τους) 3,5,3,6,5,1 τότε ο παίκτης με id 3, που έριξε 6, θα είναι σίγουρα ο πρώτος και ο παίκτης με id 5, που έριξε 1, θα είναι σίγουρα ο τελευταίος. Οπότε, παλεύουν για την 2η και 3η θέση οι παίκτες με id 1 και 4, που έριξαν 5 και για την 4η και 5η θέση, παλεύουν οι παίκτες με id 0 και 2, που έριξαν 3. Οπότε αυτοί οι παίκτες ανά δύο ξαναρίχνουν. Οπότε ξαναγίνεται ο ίδιος έλεγχος αναδρομικά μέχρι ο παίκτης του κάθε ζευγαριού να έχει μοναδική ζαριά από το ζευγάρι του.
* Ορίζουμε ένα δυσδιάστατο πίνακα array 6x6. Μέσω μίας for, η οποία γίνεται τόσες φορές όσοι είναι και οι παίκτες, δίνονται (μέσω της συνάρτησης Math.random) τυχαίες τιμές από 1 έως 6, που είναι και οι πιθανές τιμές για ένα ζάρι, στη μεταβλητή temp. Αυτές οι τιμές αποθηκεύονται στον hashmap first σαν value με κλειδιά τα αντίστοιχα i που είναι τα id-1 των παικτών. Αυτές οι τιμές όμως, αποθηκεύονται και στον πίνακα “array” με τον εξής τρόπο: το id του παίκτη αποθηκεύεται-συμπίπτει με τον αριθμό των γραμμών-1 και ο αριθμός του ζαριού που έριξε ο παίκτης, αποθηκεύεται-συμπίπτει με τον αριθμό της στήλης του πίνακα -1, καθώς η αρίθμηση στον πίνακα είναι από 0 έως 5, αλλά στο ζάρι από 1 έως 6 .Έτσι έχουμε και το id του παίκτη και τον αριθμό της πρώτης ζαριάς που έριξε σε έναν πίνακα, μέσω των διαστάσεων του και στο συγκεκριμένο στοιχείο κάθε φορά εκχωρείται σαν τιμή ένας άσσος. Για παράδειγμα αν ο παίκτης με “id 2” ρίξει 4 τότε “array[1]][3] = 1”.
* Μετά γίνεται κλίση της “checkTable” με όρισμα τον πίνακα “array”, η οποία αποθηκεύεται σε μια μεταβλητή “rowsToCheck” τύπου “ArrayList’. Όσον αφορά για την “checkTable” :
* Έχει σαν όρισμα έναν δεδομένο, δύο διαστάσεων, πίνακα κάθε φορά, (ξεκινώντας με τον “array”).
* Δημιουργείται μια μεταβλητή “rowsToCheck” τύπου “ArrayList” με όρισμα “integer” και ένας πίνακας check τύπου “int” μονοδιάστατος.
* Μέσω δύο “for” που γίνονται 6 φορές δίνονται τιμές στον πίνακα check. Πρακτικά σε κάθε “i” στοιχείο του πίνακα check προστίθενται όλα τα στοιχεία των “i” στηλών του δυσδιάστατου πίνακα 1. Οπότε αν υπάρχουν άσσοι και άρα ζαριές σε μία στήλη τότε το “check[i]” αυξάνεται.
* Μέσω μιας δεύτερης “for” ελέγχεται αν το κάθε στοιχείο του “check” είναι πάνω από 1 και άρα αν υπάρχουν πάνω από ένας παίκτης που να έφερε μια συγκεκριμένη ζαριά. Αν υπάρχουν αυτό το “i” προστίθεται στη μεταβλητή “rowsToCheck", η οποία επιστρέφεται από τη συνάρτηση αυτή, ώστε μετά να μελετηθεί η στήλη αυτή του πίνακα από τη συνάρτηση “checkRow”.
* Ακολουθεί μια for για όσες είναι οι “rowsToCheck”, που προέκυψαν από την “checkTable”, στην οποία καλείται η “checkRow" με ορίσματα όλα τα “rowsToCheck”, το “iter” σαν 0, καθώς είναι η πρώτη κλίση της συνάρτησης αυτής, μετά αν υπάρξει κοινή ζαριά θα αλλάξει, (πρακτικά είναι ο αριθμός των φορών που ξαναρίχνεται το ζάρι σε κάποια ομάδα παικτών που έχουν φέρει το ίδιο) και ο πίνακας “array”, επίσης γιατί αυτή είναι η πρώτη κλίση της συνάρτησης και αν υπάρξει κοινή ζαριά θα αλλάξει και πάλι και θα έχει σαν όρισμα έναν δεδομένο πίνακα. Όσον αφορά τη “checkRow”:
* Μέσω δυο “for” αντιγράφεται ο δεδομένος πίνακας του ορίσματος στον πίνακα “newTable”.
* Με μία “for” μηδενίζονται όλες οι στήλες του πίνακα εκτός από αυτήν που είναι το “rowsToCheck”, δηλαδή το “row”, τη δεδομένη φορά, ώστε να τη μελετήσουμε καλύτερα.
* Στην επόμενη “for” προστίθενται όλα τα στοιχεία της συγκεκριμένης στήλης(“row”) σε μία μεταβλητή check. Οπότε αν το στοιχείο είναι 1 σημαίνει ότι υπάρχει ζαριά έτσι μηδενίζεται, αφού η τιμή του έχει δοθεί ήδη στο “check” και άρα έχει μελετηθεί ήδη και δίνεται στο συγκεκριμένο παίκτη μια καινούργια ζαριά σε κάποιο σημείο του πίνακα, ξανά βάζοντας με αντίστοιχο τρόπο στο στοιχείο αυτό τον άσσο (σε περίπτωση που χρειαστεί να κληθεί ξανά η “checkTable”).
* Αυτά όμως τα στοιχεία, οι νέες ζαριές δηλαδή, χρησιμοποιούνται μόνο αν το “check” είναι μεγαλύτερο του 1 δηλαδή μόνο αν και κάποιος ή κάποιοι άλλοι παίκτες σε αυτό το “iter” έχουν φέρει την ίδια ζαριά. Τότε το “iter” μειώνεται κατά 1 και αν είναι μικρότερο από “min\_iter” αποθηκεύεται σε αυτό, ώστε στο τέλος που θα γίνει ο πολλαπλασιασμός που θα εξηγηθεί παρακάτω να πάρει πολλαπλασιαστεί με το μέγιστο εκθέτη για να μη χαθεί κάποιο δεκαδικό ψηφίο.
* Μέσω δύο ακόμη “for” αν πάντα το “check” είναι μεγαλύτερο του 1 αποθηκεύονται οι καινούργιες ζαριές των παικτών στο χάρτη “first”, κρατώντας την αρχική ζαριά σαν ακέραιο και προσθέτοντας κάθε φορά σαν δεκαδικό ψηφίο την επόμενη.
* Τέλος αν το “check” είναι μεγαλύτερο του 1 ξανακαλώ τη “checkTable”, με τον καινούργιο πίνακα που έχει δημιουργηθεί, και αποθηκεύω αυτό που επιστρέφει στο “rowsToCheck”, μετά ξανακαλώ την “checkrow” με το καινούργιο “iter”, το αντίστοιχο “rowCheck” και ούτω καθεξής.
  + Συνεχίζοντας τη “setTurns” δημιουργούμε ένα “LinkedHashMap” με όνομα “sorted” και ορίσματα “integer,integer” και μέσω μιας “do while” για όσο το “counter” είναι μικρότερο του αριθμού των παικτών, ταξινομούμε τον χάρτη “first”. Οπότε μέσα στη “do while” μέσω μιας “for” βρίσκουμε το ελάχιστο από όλα τα στοιχεία του χάρτη “first”, παίρνουμε το κλειδί του και το ορίζουμε κλειδί του “sorted” και ως “value” της “sorted” ορίζουμε την στρογγυλοποίηση της τιμής του κλειδιού του αυτού του “first”, πολλαπλασιασμένη επί 10^(-iter), ώστε να φαίνεται σαν μια σειρά αριθμών όλες οι ζαριές που έχει ρίξει ο κάθε παίκτης. Τέλος αυξάνεται ο “counter” και η τιμή του κλειδιού αυτού του “first” γίνεται 10 ώστε να μη μπορεί να είναι ποτέ ξανά το ελάχιστο και να βρει το αμέσως επόμενο μικρότερο στοιχείο στην επομένη επανάληψη. Τέλος επιστρέφεται ο ταξινομημένος αυτός χάρτης.
* Μία συνάρτηση “main”.
* Ορίζουμε τις μεταβλητές “id1”, “id2”, “ro” τύπου “int” για να αρχικοποιούμε τον αριθμό του πλακιδίου που βρίσκεται κάθε παίκτης και τον αριθμό των γύρων του παιχνιδιού και δημιουργούμε και έναν πίνακα “array” με διαστάσεις [2][5].
* Δημιουργώ ένα αντικείμενο “g1” τύπου “Game” με όρισμα 0, ένα αντικείμενο “b3” τύπου “Board” με ορίσματα (20,10,3,3,6) και μέσω αυτού καλώ την συνάρτηση “createBoard” και την “createElementBoard” για να δημιουργηθούν οι κατάλληλοι πίνακες. Επίσης δημιουργώ δύο αντικείμενα ένα “p1” τύπου “Player” με ορίσματα (1, "maria",0, b3) και ένα “p2” τύπου “HeuristicPlayer” με ορίσματα (2, "mhtsos", 0, b3, ro) αντίστοιχα.
* Αρχικοποιούνται οι γύροι, προσθέτω τους δύο παίκτες στο “arrayList” “players” και καλώ τη συνάρτηση “SetTurns”.
* Σε μία “do while”, μέχρι κάποιος από τους δύο παίκτες να τερματίσει ή οι γύροι να γίνουν πάνω από 100, ορίζεται η σειρά που θα παίζουν οι παίκτες, χρησιμοποιώντας το ταξινομημένο “map” σε μία “for”, οπότε αν συναντάει “Heuristic Player” καλεί τη “getNextMove”, ενώ αν συναντά “Player” καλεί τη move.
* Tέλος εκτυπώνονται ο νικητής ανάλογα με τη συνάρτηση στόχου και τους πόντους, οι πόντοι, οι γύροι και τα στατιστικά του “HeuristicPlayer”.