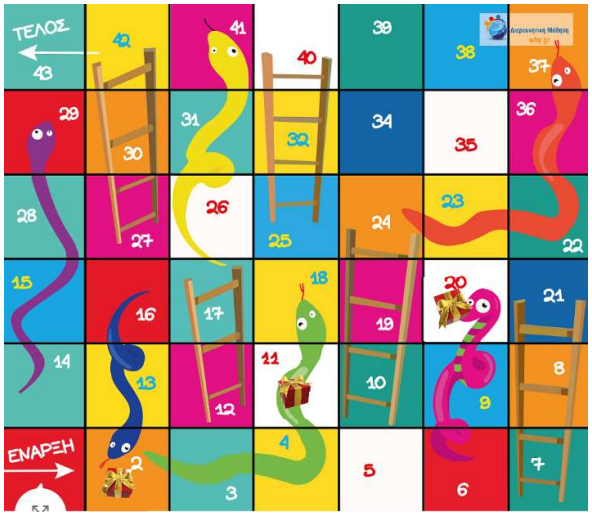
ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Το Φιδάκι

Το φιδάκι είναι ένα επιτραπέζιο παιχνίδι, το οποίο παίζεται σε ένα ταμπλό χτισμένο σε επίπεδα, τα οποία ανεβαίνεις εάν πέσεις σε βάση σκάλας και κατεβαίνεις εάν πέσεις σε στόμα από φιδάκι και παίρνεις το δώρο αν πέσεις σε πλακίδιο με δώρο. Το παιχνίδι παίζεται με 2 παίκτες και στόχος είναι να φτάσουν στο τελευταίο πλακίδιο του ταμπλό αποφεύγοντας τα εμπόδια.

Στην αρχή του παιχνιδιού, όλοι οι παίκτες ξεκινάνε από την έναρξη και προσπαθούν να καταλήξουν στο τέρμα. Ρίχνουν το ζάρι και όποιος ρίξει το μικρότερο αριθμό παίζει πρώτος, εκείνος που έχει το δεύτερο μικρότερο αριθμό παίζει δεύτερος κ.ο.κ. Ο πρώτος παίκτης ρίχνει ξανά το ζάρι και προχωράει όσα βήματα του δείχνει αυτό. Εάν φτάσει σε κάποιο από τα πλακίδια που του δείχνει ότι ξεκινάει η σκάλα, την ανεβαίνει. Η κάθε σκάλα μπορεί να χρησιμοποιηθεί μόνο μια φορά κατά τη διάρκεια του παιχνιδιού. Αυτό σημαίνει ότι όταν ένας παίκτης ανέβει μια σκάλα, στη συνέχεια η σκάλα σπάει και δε μπορεί να ξαναχρησιμοποιηθεί. Αν όμως φτάσει σε κάποιο σημείο του ταμπλό στο οποίο βρίσκεται το κεφάλι από ένα φιδάκι, κατεβαίνει στο αντίστοιχο πλακίδιο που τελειώνει η ουρά του. Οι παίκτες παίζουν κυκλικά σύμφωνα με τη σειρά που έχει οριστεί στην αρχή του παιχνιδιού, δηλαδή όταν τελειώσει ο πρώτος παίκτης τη σειρά του, συνεχίζει ο δεύτερος, στη συνέχεια ο τρίτος κ.ο.κ.



Εικόνα 1: Παράδειγμα ταμπλό παιχνιδιού διάστασης 7x6.

Εκτός όμως από τα φιδάκια και από τις σκάλες, ένα πλακίδιο μπορεί να περιέχει ένα δώρο, το οποίο. Όταν ένας παίκτης πάρει ένα δώρο αυτό τότε αυτό εξαφανίζεται από το ταμπλό.

Νικητής του παιχνιδιού είναι ο παίκτης που θα καταφέρει να φτάσει στο τελευταίο πλακίδιο του ταμπλό όσο το δυνατόν γρηγορότερα και με το μεγαλύτερο δυνατό σκορ.

Εργασία Β – Heuristic Algorithm (0,5 βαθμος)

Σε αυτή την εργασία καλείστε να υλοποιήσετε έναν παίκτη HeuristicPlayer ο οποίος έχει τη δυνατότητα να **ορίζει ο ίδιος το ζάρι** που χρειάζεται σε κάθε γύρο για να κάνει τη βέλτιστη δυνατή κίνηση. Ζητείται λοιπόν να δημιουργήσετε μια **συνάρτηση αξιολόγησης των διαθέσιμων κινήσεων** που έχει ο παίκτης σε κάθε γύρο παιχνιδιού και την αντίστοιχη **συνάρτηση επιλογής της καλύτερης κίνησης**. Σκοπός σας είναι να δημιουργήσετε μια όσο το δυνατόν πιο ολοκληρωμένη συνάρτηση αξιολόγησης, που να λαμβάνει υπόψη της τα διαθέσιμα δεδομένα (μέχρι και) εκείνη τη στιγμή και να τα αξιολογεί κατάλληλα, εφαρμόζοντας διάφορα κριτήρια, με στόχο να κάνει τον παίκτη να προηγείται στο ταμπλό κάνοντας σε κάθε γύρο τα περισσότερα δυνατά βήματα (steps), και κερδίζοντας τους περισσότερους δυνατούς πόντους (gainPoints). Η συνάρτηση στόχου που θα βελτιστοποιήσετε θα οριστεί από εσάς. Ένα παράδειγμα συνάρτησης στόχου είναι η παρακάτω:

$$\text{f(steps, gainPoints)} = \text{steps} * 0{,}65 + \text{gainPoints} * 0{,}35$$

Ένα παράδειγμα εξέτασης της συγκεκριμένης συνάρτησης στόχου για το ταμπλό της Εικόνας 1 για τον παίκτη που είναι στη θέση 6 περιγράφεται παρακάτω:

Ο παίκτης έχει στη διάθεσή του 6 δυνατές κινήσεις.

- *Στην περίπτωση που επιλέξει το ζάρι του να έχει τον αριθμό **1**, θα κάνει 15 βήματα και 0 πόντους του δώρου->* $\text{f(steps, gainPoints)} = 15*\mathbf{0{,}65=}9{,}75$
- *Στην περίπτωση που επιλέξει το ζάρι του να έχει τον αριθμό **2**, θα κάνει **2** βήματα και θα κερδίσει **0** πόντους ->* $\text{f(steps, gainPoints)} = 2 * 0{,}65 + 0 * 0{,}35 = \mathbf{1{,}3}$.
- *Στην περίπτωση που επιλέξει το ζάρι του να έχει τον αριθμό **3**, θα κάνει **3** βήματα και θα κερδίσει **0** πόντους ->* $\text{f(steps, gainPoints)} = \mathbf{3*0{,}65+0*0{,}35}.=1{,}95$
- *Στην περίπτωση που επιλέξει το ζάρι του να έχει τον αριθμό **4**, θα κάνει **18** βήματα και θα κερδίσει **0** πόντους ->* $\text{f(steps, gainPoints)} = \mathbf{18*0{,}65+0*0{,}35}. = \mathbf{11{,}7}$
- *Στην περίπτωση που επιλέξει το ζάρι του να έχει τον αριθμό **5**, θα κάνει **5** βήματα και θα κερδίσει **10** πόντους από το δώρο->* $\text{f(steps, gainPoints)} = \mathbf{5*0{,}65+10*0{,}35=7}$.
- *Στην περίπτωση που επιλέξει το ζάρι του να έχει τον αριθμό **6**, θα κάνει **11** βήματα και θα κερδίσει **0** πόντους ->* $\text{f(steps, gainPoints)} = \mathbf{11*0{,}65+0*0{,}35}.=7{,}15.$

Ο κώδικας θα συμπληρωθεί από εσάς στον κώδικα που ετοιμάσατε για πρώτο κομμάτι της εργασίας. Στη συνέχεια παρουσιάζεται η νέα κλάση **HeuristicPlayer** που σας ζητείται να υλοποιήσετε, καθώς και η κλάση **Game** όπου θα προσθέσετε επιπλέον λειτουργικότητα.

Κλάση HeuristicPlayer

Η κλάση **HeuristicPlayer** θα αντιπροσωπεύει τον παίκτη που παίζει με στρατηγική. Κληρονομεί την κλάση **Player** και έχει τις εξής επιπλέον μεταβλητές:

i. **`ArrayList<Integer[]> path`:** πληροφορίες για την κάθε κίνηση του παίκτη κατά τη διάρκεια του παιχνιδιού. Πιο συγκεκριμένα, η δομή θα περιλαμβάνει πληροφορίες για το ζάρι, τον αριθμό των πόντων που προσέφερε η κίνηση, τον αριθμό των βημάτων του, τον αριθμό των δώρων που πήρε, τον αριθμό των φιδιών από τα οποία τσιμπήθηκε και τον αριθμό των σκαλών που χρησιμοποίησε ο παίκτης στη συγκεκριμένη κίνησή του. Μπορείτε να αποθηκεύσετε και ό,τι άλλο μπορεί να σας φανεί χρήσιμο.

Οι συναρτήσεις που πρέπει να υλοποιήσετε είναι οι εξής:

a. Οι **constructors της κλάσης**.
b. Συνάρτηση **`double evaluate(int currentPos, int dice)`:** Η συνάρτηση αυτή 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 έχετε ορίσει.
c. Συνάρτηση **`int getNextMove(int currentPos):`** Η συνάρτηση αυτή είναι υπεύθυνη για την επιλογή της τελικής κίνησης του παίκτη. Η συνάρτηση θα πρέπει να περιλαμβάνει τα παρακάτω:
    i. Δημιουργία μιας δομής (Δυναμικός ή μη πίνακας) που θα αποθηκεύει τις πιθανές κινήσεις του παίκτη καθώς και την αξιολόγηση κάθε μιας από αυτές.
    ii. Αξιολόγηση κάθε πιθανής κίνησης με χρήση της συνάρτησης `double evaluate(int currentPos, int dice)` και αποθήκευση της αξιολόγησης στη δομή που έχετε φτιάξει.
    iii. Επιλογή της καλύτερης κίνησης με βάση την αξιολόγηση που κάνατε.
    iv. Μετακίνηση του παίκτη.
    v. Ανανέωση της μεταβλητής της κλάσης `path`.
    vi. Επιστροφή της νέας θέσης του παίκτη.

    ΠΡΟΣΟΧΗ!!! **Κατά τη διάρκεια των δοκιμών που θα κάνετε για την επιλογή της βέλτιστης κίνησης, θα πρέπει να προσέξετε ώστε να αφήσετε αναλλοίωτο το ταμπλό και το σκορ των παικτών. Αυτά θα πρέπει να αλλάξουν μόνο μετά την επιλογή της βέλτιστης κίνησης, δηλαδή κατά τη διάρκεια της πραγματικής μετακίνησης του παίκτη.**

d. Συνάρτηση **`void statistics()`**: Η συνάρτηση αυτή εκτυπώνει στοιχεία για τις κινήσεις του παίκτη σε κάθε γύρο του παιχνιδιού (ζάρι που επέλεξε και ενέργειες που έκανε πχ «ο παίκτης στο round 1 έθεσε το ζάρι ίσο με 4 και ανέβηκε μια σκάλα»), καθώς και στατιστικά στοιχεία για το σύνολο των κινήσεών του. Συγκεκριμένα, ζητείται να εκτυπώνονται:
    - Ο αριθμός επισκέψεων πλακιδίου που περιέχει κεφάλι φιδιού.
    - Ο αριθμός επισκέψεων πλακιδίου που περιέχει βάση σκάλας.

- Ο αριθμός επισκέψεων πλακιδίου που περιέχει δώρο.

Κλάση Game

Η κλάση **Game** θα αντιπροσωπεύει το παιχνίδι και θα έχει τις εξής μεταβλητές:

i. **`int round:`** ο τρέχον γύρος του παιχνιδιού.

Οι συναρτήσεις που πρέπει να υλοποιήσετε είναι οι εξής:

a. Οι **constructors της κλάσης**.
b. Όλες οι συναρτήσεις ***get*** και ***set*** για τη μεταβλητή της κλάσης.
c. Συνάρτηση **`Map<Integer,Integer> setTurns(ArrayList<Object> players)`**: συνάρτηση καθορισμού της σειράς με την οποία θα παίζουν οι παίκτες. Στην αρχή του παιχνιδιού οι παίκτες ρίχνουν το ζάρι και όποιος ρίξει το **μικρότερο** αριθμό παίζει **πρώτος**, εκείνος που έχει το δεύτερο μικρότερο αριθμό παίζει δεύτερος κ.ο.κ. Η συνάρτηση επιστρέφει ένα χάρτη με τα ids των παικτών και τη ζαριά που έφεραν, ταξινομημένα με βάση τη ζαριά.
d. Συνάρτηση **`public static void main(String[] args)`**: συνάρτηση εκκίνησης του παιχνιδιού. Στη συνάρτηση αυτή θα πρέπει να γίνεται μια ακολουθία ενεργειών:
    - Δημιουργία ταμπλό διάστασης 10x20, με 3 φίδια, 3 σκάλες και 6 δώρα.
    - Ορισμός 2 παικτών, ένας παίκτης που παίζει με τυχαίες κινήσεις και ένας παίκτης τύπου HeuristicPlayer.
    - Καθορισμός της σειράς με την οποία θα παίζουν οι παίκτες.
    - Οι παίκτες παίζουν εναλλάξ μέχρις ότου φτάσουν σε ένα συγκεκριμένο γύρο πχ round 100 ή κάποιος τερματίσει. Τότε το παιχνίδι τελειώνει. Ο νικητής είναι αυτός που στο τέλος του παιχνιδιού προηγείται του συμπαίχτη του στο ταμπλό και ταυτόχρονα έχει το μεγαλύτερο σκορ. Εδώ μπορείτε να χρησιμοποιήσετε τα βάρη που θέσατε στη συνάρτηση στόχου για τις δύο παραμέτρους, ώστε να καθορίσετε το νικητή του παιχνιδιού. Σε περίπτωση ισοπαλίας, νικητής είναι αυτός που τερμάτισε πρώτος.
    - Εκτύπωση στατιστικών στοιχείων για τον παίκτη τύπου HeuristicPlayer.
    - Εκτύπωση του αριθμού των γύρων παιχνιδιού που έπαιξαν οι παίκτες, του σκορ του κάθε παίκτη και το νικητή του παιχνιδιού.

ΠΡΟΣΟΧΗ!!!

- **Αν θέλετε να διαβάσετε τις τιμές των μεταβλητών ενός αντικειμένου μιας κλάσης ή να θέσετε τιμές στις μεταβλητές θα πρέπει να χρησιμοποιήσετε τους αντίστοιχους getters και setters.**
- **Μπορείτε να προσθέσετε επιπλέον μεταβλητές και συναρτήσεις στις κλάσεις εφόσον αυτό σας εξυπηρετεί.**

Οδηγίες

Τα προγράμματα θα πρέπει να υλοποιηθούν σε Java. Επίσης, πριν από κάθε κλάση ή μέθοδο θα υπάρχει επικεφαλίδα σε μορφή σχολίων με σύντομη περιγραφή της λειτουργικότητας του κώδικα. Στην περίπτωση των μεθόδων, πρέπει να περιγράφονται και οι μεταβλητές τους.

Οι εργασίες που περιέχουν λάθη μεταγλώττισης θα μηδενίζονται αυτομάτως.

Παραδοτέα:

1. **Ένα αρχείο σε μορφή .zip με όνομα "AEM_PartB.zip",** το οποίο θα περιέχει **όλο** το project σας στον eclipse. Το αρχείο .zip θα γίνεται upload στο site του μαθήματος ***στην ενότητα των ομαδικών εργασιών και μόνο***. Τα ονόματα των αρχείων πρέπει να είναι με **λατινικούς χαρακτήρες**.

Προθεσμία υποβολής:

Πέμπτη 12 Ιανουαρίου, 23:59 (ηλεκτρονικά)
Δε θα υπάρξει καμία παρέκκλιση από την παραπάνω προθεσμία.