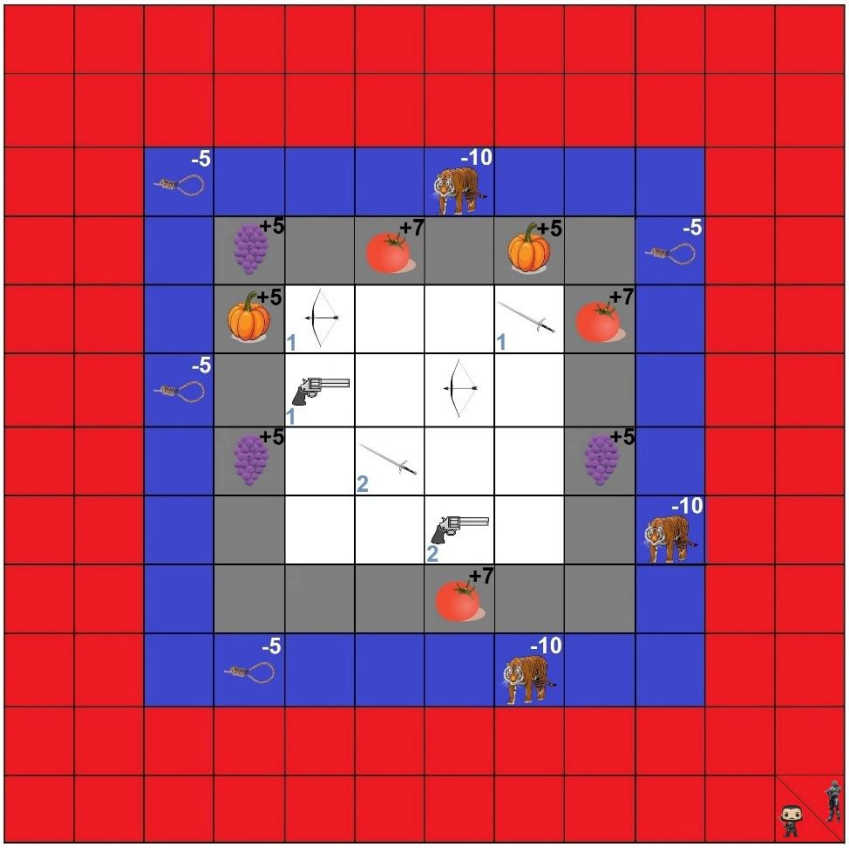
Εργαστήριο Επεξεργασίας Πληροφορίας και Υπολογισμών
Τομέας Ηλεκτρονικής και Υπολογιστών
ΤΗΜΜΥ Α.Π.Θ

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Hunger Games

May the odds be in your favor..

Το Hunger Games είναι ένα τηλεοπτικό ετήσιο γεγονός στο οποίο οι συμμετέχοντες είναι αναγκασμένοι να αγωνιστούν μέχρι θανάτου, έως ότου να υπάρξει κάποιος νικητής. Στην απλουστευμένη παραλλαγή του παιχνιδιού που καλείστε να υλοποιήσετε, συμμετέχουν 2 παίκτες, οι οποίοι παίζουν εναλλάξ και μετακινούνται κατά μία θέση πάνω στο ταμπλό κάθε φορά, έχοντας σαν στόχο να επιβιώσουν και να μαζέψουν πόντους. Ο ένας παίκτης μπορεί να σκοτώσει τον άλλον αν αυτός βρίσκεται σε κοντινή απόσταση και διαθέτει το κατάλληλο όπλο (πιστόλι). Νικητής του παιχνιδιού είναι εκείνος που μείνει ζωντανός ή, σε περίπτωση που το παιχνίδι τερματίσει επειδή το ταμπλό συρρικνώθηκε, εκείνος που έχει το υψηλότερο σκορ.



Εικόνα 1: Παράδειγμα ταμπλό παιχνιδιού διάστασης 12x12.

Εργασία C – MinMax Algorithm (0,5 βαθμοί)

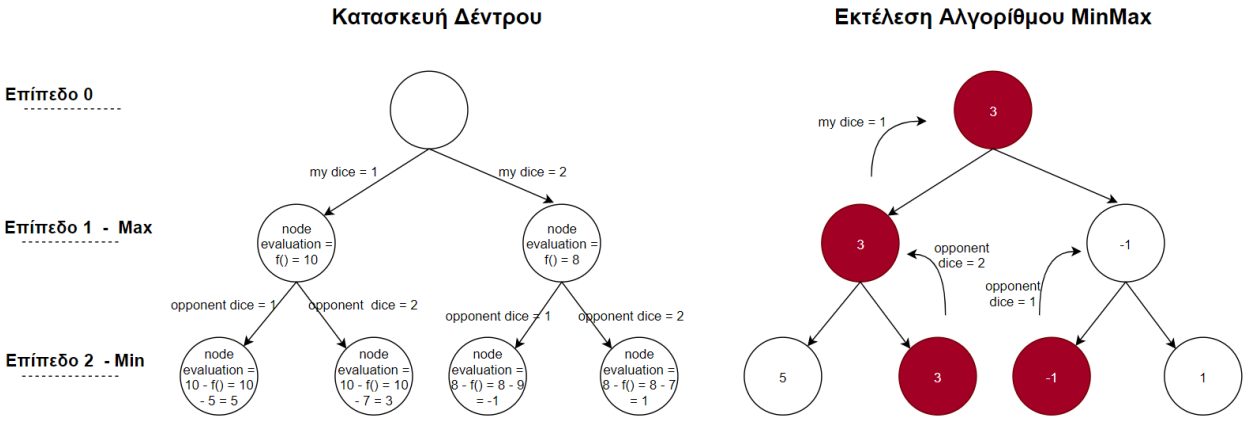
Στο τρίτο παραδοτέο καλείστε να υλοποιήσετε τον αλγόριθμο MinMax για τη βελτιστοποίηση του παίκτη σας. Σκοπός σας είναι να δημιουργήσετε ένα δέντρο βάθους 2 κινήσεων, το όποιο, σε συνδυασμό με τη συνάρτηση αξιολόγησης που είχατε δημιουργήσει στην δεύτερη εργασία (ή μια βελτιωμένη έκδοση αυτής), θα αξιολογεί τις διαθέσιμές κινήσεις σε βάθος χρόνου και θα επιλέγει την καλύτερη δυνατή για τον επόμενο γύρο.

Αλγόριθμος MinMax

Όπως σε πολλά παιχνίδια, έτσι και στο δικό μας, για τον υπολογισμό της βέλτιστης κίνησης δεν είναι εφικτό να υπολογίσουμε όλες τις διαθέσιμες κινήσεις (μέχρι το τέλος του παιχνιδιού) σε εύλογο χρονικό διάστημα. Εξαιτίας αυτού του γεγονότος, το δένδρο των πιθανών συνδυασμών αναπτύσσεται μέχρι ενός βάθους (π.χ. για βάθος 2 κινήσεων έχω όλες τις δυνατές κινήσεις μου και όλες τις πιθανές κινήσεις-απαντήσεις του αντιπάλου). Σε κάθε φύλλο του δένδρου εξετάζω τη συνάρτηση αξιολόγησης τόσο του δικού μου παίκτη όσο και του αντιπάλου (υπολογίζω τη διαφορά $f_{max\_node} - f_{min\_node}$).

Το δένδρο που θα φτιαχτεί θα έχει τη δομή της Εικόνας 2, προσαρμοσμένο έτσι ώστε ο κάθε κόμβος να έχει τόσα παιδιά όσες και οι διαθέσιμες κινήσεις, δηλαδή στη γενική περίπτωση που κανένας από τους δύο παίκτες δε βρίσκεται στα σύνορα του ταμπλό, οι διαθέσιμες κινήσεις είναι 8.

Θεωρούμε πάντα ότι ο αντίπαλος θα παίξει τη χειρότερη για εμάς κίνηση. Για αυτό επιλέγει τον κόμβο με την **ελάχιστη (Minimum)** τιμή της διαφοράς $f_{max\_node} - f_{min\_node}$ (επίπεδο 2). Εμείς φυσικά θα παίξουμε την κίνηση που **μεγιστοποιεί (Maximum)** αυτή τη διαφορά (επίπεδο 1), λαμβάνοντας ως δεδομένο ότι ο αντίπαλος θα παίξει την καλύτερη κίνηση για αυτόν. Στην Εικόνα 2 για παράδειγμα, εάν εμείς παίξουμε το ζάρι 1 στο Επίπεδο 1, τότε θεωρούμε ότι ο αντίπαλος θα παίξει το ζάρι 2 στο Επίπεδο 2 (γιατί $3 < 5$), ενώ εάν επιλέξουμε το ζάρι 2 τότε ο αντίπαλος θα παίξει το ζάρι 1 (γιατί $-1 < 1$). Τελικά, θα επιλέξουμε να παίξουμε το ζάρι 1 (Επίπεδο 1), γιατί $3 > -1$.



Εικόνα 2: Κατασκευή δέντρου βάθος 2 κινήσεων και εκτέλεση του αλγορίθμου MinMax.

Παρακάτω περιγράφονται οι κλάσεις που θα πρέπει να υλοποιήσετε σε αυτό το παραδοτέο.

Κλάση Node

Για την υλοποίηση ενός δέντρου διαθέσιμων κινήσεων προτείνεται η δημιουργία μιας κλάσης με το όνομα Node. Η κλάση αυτή θα έχει ως μεταβλητές:

i. **Node parent**: ο κόμβος-πατέρας του κόμβου που δημιουργήσατε.
ii. **ArrayList<Node> children**: ο δυναμικός πίνακας που περιλαμβάνει τα παιδιά του κόμβου που δημιουργήσατε.
iii. **int nodeDepth**: το βάθος του κόμβου στο δέντρο του MinMax Αλγορίθμου.
iv. **int[] nodeMove**: την κίνηση που αντιπροσωπεύει το Node, σαν πίνακας ακεραίων που περιλαμβάνει το x, y του πλακιδίου της τρέχουσας θέσης και τον αριθμό του ζαριού.
v. **Board nodeBoard**: το ταμπλό του παιχνιδιού για το συγκεκριμένο κόμβο-κίνηση.
vi. **double nodeEvaluation**: την τιμή της συνάρτησης αξιολόγησης της κίνησης που αντιπροσωπεύει ο συγκεκριμένος κόμβος.

Οι συναρτήσεις της κλάσης που θα πρέπει να υλοποιηθούν είναι:

a. **Node ():** Ένας ή περισσότεροι constructors για την κλάση σας, με διαφορετικά ορίσματα.
b. Κατάλληλες συναρτήσεις ***get*** και ***set***.

Κλάση MinMaxPlayer

Η κλάση **MinMaxPlayer** θα αντιπροσωπεύει τον παίκτη που παίζει χρησιμοποιώντας τον αλγόριθμο MinMax. Η κλάση αυτή θα είναι παρόμοια με την κλάση **HeuristicPlayer** που είχατε υλοποιήσει στο δεύτερο παραδοτέο. Όπως η **HeuristicPlayer**, έτσι και η **MinMaxPlayer** θα κληρονομεί την κλάση **Player** και θα έχει όλες τις μεταβλητές και τις συναρτήσεις που είχε και η **HeuristicPlayer** τροποποιημένες κατάλληλα στις ανάγκες αυτού του παίκτη **MinMaxPlayer**. Επιπλέον, θα περιέχει τις συναρτήσεις που απαιτούνται για τη δημιουργία του δέντρου αξιολόγησης με σκοπό την επιλογή της κίνησης σύμφωνα με τον παρακάτω αλγόριθμο.

Οι νέες συναρτήσεις που πρέπει να υλοποιήσετε είναι οι εξής:

a. **`double evaluate(int dice, int x, int y, Player opponent)`**: Η συνάρτηση αυτή αξιολογεί την κίνηση του παίκτη όταν αυτός έχει τη ζαριά dice, δεδομένου ότι βρίσκεται στη θέση `x,y`. Η συνάρτηση επιστρέφει την αξιολόγηση της κίνησης, σύμφωνα με τη συνάρτηση στόχου που έχετε ορίσει. Το όρισμα `opponent` αποτελεί τον αντίπαλο του παίκτη για τον οποίο καλείται η συνάρτηση `evaluate()`. Εδώ μπορείτε να χρησιμοποιήσετε τη συνάρτηση που

υλοποιήσατε στη δεύτερη εργασία (τροποποιημένη κατάλληλα ώστε να δέχεται και να αξιοποιεί τα νέα ορίσματα) ή μια βελτιωμένη έκδοση αυτής.

b. **`int chooseMinMaxMove(Node root)`**: Η συνάρτηση αυτή υλοποιεί τον MinMax αλγόριθμο για να βρει τη βέλτιστη διαθέσιμη κίνηση και επιστρέφει το δείκτη της καλύτερης κίνησης.
c. Τις συναρτήσεις δημιουργίας του δέντρου, όπως περιγράφονται παρακάτω.

Αλγόριθμος Δημιουργίας Δέντρου για βάθος 2 κινήσεων

```
int[] getNextMove (int xCurrentPos, int yCurrentPos, int xOpponentCurrentPos, int
yOpponentCurrentPos)
  Use current board to create a new node which corresponds to the root of the tree.
  Call createMySubtree(root, 1,..)
  // The tree is now finished
  Call the chooseMinMaxMove(Node root)to choose the best available move.
  Move the player.
  Update path variable.
  Return the best move.

void createMySubtree(Node root, int depth, int xCurrentPos, int yCurrentPos, int
xOpponentCurrentPos, int yOpponentCurrentPos)
  Find the number of available movements.
  For each available movement (dice):
        Create a clone of the root node's board and simulate making the movement.
        Create a new node as child of the parent node using the new board state.
        child.nodeEvaluation <- evaluate(dice, ..)
        Add the node (newNode) as child of the parent node.
        Complete the tree branches by calling createOpponentSubtree(newNode, depth+1,
        ..)

void createOpponentSubtree(Node parent, int depth, int xCurrentPos, int yCurrentPos,
int xOpponentCurrentPos, int yOpponentCurrentPos)
  Find the number of available movements of this new state of the board.
  For the number of available movements for the opponent's turn (dice):
         Create a clone of the parent node's board and simulate making the movement.
         Create a new node as child of the parent node using the new board state.
         child.nodeEvaluation <- parent.nodeEvaluation - evaluate(dice, ..)
         Add the node as child of the parent node.
```

Προσοχή!!!! Κατά τη δημιουργία του δέντρου για την αξιολόγηση των πιθανών κινήσεων του παίκτη σας και του αντιπάλου, θα πρέπει να προσέχετε ώστε να διατηρούνται το αρχικό ταμπλό, οι συντεταγμένες των παικτών και οι πόντοι των παικτών αναλλοίωτοι. Αυτό σημαίνει ότι θα πρέπει να δημιουργούνται κλώνοι του αρχικού board κατά τη δημιουργία του δέντρου, δηλαδή κατά την κλήση των συναρτήσεων `createMySubTree()`, `createOpponentSubtree()`.

Κλάση Game

Στην κλάση **Game** που υλοποιήσατε στο δεύτερο παραδοτέο, θα πρέπει να αντικαταστήσετε τον παίκτη **HeuristicPlayer** με τον παίκτη **MinMaxPlayer**. Παρακάτω σημειώνονται κάποιες τροποποιήσεις που πρέπει να γίνουν σε σχέση με το 2ο παραδοτέο.

1. Οι παίκτες έχουν το ίδιο σημείο εκκίνησης, το οποίο είναι το γωνιακό πλακίδιο κάτω δεξιά, όπως φαίνεται στην Εικόνα 1.

2. Το αρχικό σκορ των παικτών είναι ίσο με 15. Σε περίπτωση που κάποιος παίκτης αποκτήσει αρνητικό σκορ κάποια στιγμή μέσα στο παιχνίδι, ο παίκτης πεθαίνει και το παιχνίδι τερματίζει με νικητή τον αντίπαλό του.
3. Ο παίκτης **MinMaxPlayer** βλέπει ολόκληρο το ταμπλό, σε αντίθεση με τον **HeuristicPlayer** που βλέπει μόνο ένα μέρος του.
4. Οτιδήποτε άλλο μπορεί να σας φανεί χρήσιμο στην υλοποίησή σας.

Οδηγίες

Τα προγράμματα θα πρέπει να υλοποιηθούν σε Java, με πλήρη τεκμηρίωση του κώδικα. Το πρόγραμμά σας πρέπει να περιέχει επικεφαλίδα σε μορφή σχολίων με τα στοιχεία σας (ονοματεπώνυμα, ΑΕΜ, τηλέφωνα και ηλεκτρονικές διευθύνσεις). 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.

Είναι δική σας ευθύνη η απόδειξη καλής λειτουργίας του προγράμματος.

Παραδοτέα:

1. **Ηλεκτρονική αναφορά** που θα περιέχει: εξώφυλλο, περιγραφή του προβλήματος, του αλγορίθμου και των διαδικασιών που υλοποιήσατε και τυχόν ανάλυσή τους. Σε καμία περίπτωση να μην αντιγράφεται ολόκληρος ο κώδικας μέσα στην αναφορά (εννοείται ότι εξαιρούνται τμήματα κώδικα τα οποία έχουν ως στόχο τη διευκρίνιση του αλγορίθμου).
   Προσοχή: Ορθογραφικά και συντακτικά λάθη πληρώνονται.

2. **Ένα αρχείο σε μορφή .zip με όνομα "AEM1_AEM2_PartC.zip",** το οποίο θα περιέχει **όλο** το project σας στον eclipse καθώς και το αρχείο της γραπτής αναφοράς σε pdf (**αυστηρά**). 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:

Κώδικας και αναφορά **Παρασκευή 20 Δεκεμβρίου, 18:00 μμ** (ηλεκτρονικά)
Δε θα υπάρξει καμία παρέκκλιση από την παραπάνω προθεσμία.