Θησέας και Μινώταυρος

Εργασία στις Δομές Δεδομένων, Μέρος Β

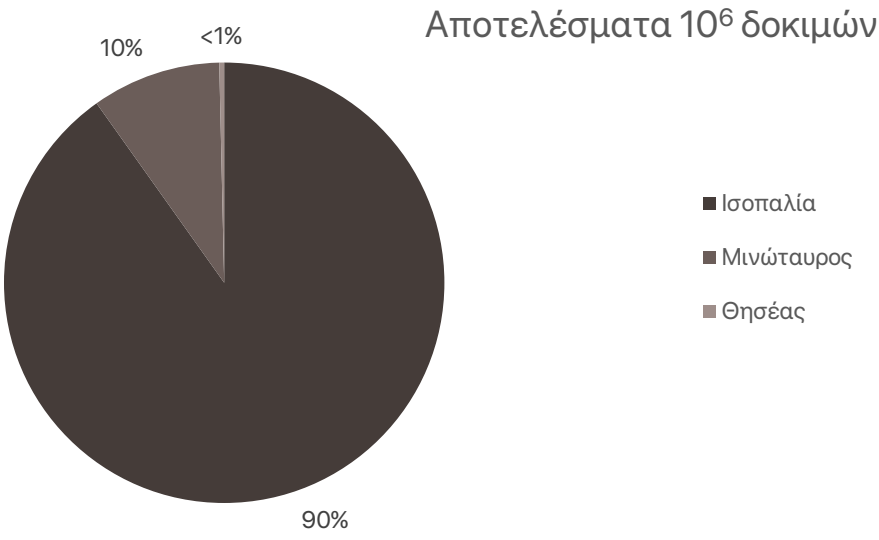
Τερζίδης Αλέξανδρος

Σουλίδης Πέτρος

Εισαγωγικό Σημείωμα

Στο πρώτο μέρος της εργασίας, υλοποιήσαμε μέσω της Java ένα παιχνίδι που προσομοιώνει τον μύθο του Θησέα και του Μινώταυρου στον Λαβύρινθο. Ο Λαβύρινθος αναπαρίσταται από ένα ταμπλό με πλακίδια και οι ήρωες από τυχαία κινούμενους παίκτες. Τοποθετούνται, επιπλέον, λάφυρα, τα οποία οφείλει να συγκεντρώσει ο Θησέας για να νικήσει τον Μινώταυρο.

Παρ' ότι η πραγματοποίηση του παραπάνω παρουσιάζει ιδιαίτερο ενδιαφέρον, οι τυχαίες κινήσεις των παικτών καθιστούν το παιχνίδι μονότονο. Εκτελώντας επανειλημμένα τον κώδικα, προέκυψαν τα αποτελέσματα του γραφήματος:



Τη μονοτονία αυτή καλούμαστε να ανακόψουμε στο δεύτερο μέρος της εργασίας. Ο Θησέας πλέον θα κινείται έξυπνα, αφού δύναται να παρατηρεί γειτονικά του πλακίδια και να αξιολογεί την κάθε του κίνηση. Είναι λογικό να επιδιώκει να κινείται προς τα λάφυρα που εντοπίζει και μακριά από το θηρίο. Καθίσταται προφανές ότι η μέθοδος από την οποία εξαρτάται η αξιολόγηση των κινήσεων διαδραματίζει καθοριστικό ρόλο στην έκβαση του παιχνιδιού.

Περιεχόμενα

Περιγραφή του Αλγορίθμου

1. Κλάση HeuristicPlayer

Πρόκειται για την κλάση που αναλαμβάνει την αναβάθμιση του απλού παίκτη που κινείται τυχαία (αντικείμενο κλάσης Player) σε ένα σχετικά ευφυή παίκτη (αντικείμενο κλάσης Player). Διακρίνουμε ότι, πέρα από τη λογική που προστίθεται στον παίκτη της HeuristicPlayer, δε διαφέρουν κάπου αλλού, επομένως η HeuristicPlayer είναι παράγωγη της Player.

Όσον αφορά τις μεταβλητές, πέρα από το δυναμικό array path που ορίζεται να προστεθεί, εισάγουμε τις βοηθητική μεταβλητή (int) ability που καθορίζει το μήκος (σε πλήθος tiles) της «όρασης» παίκτη. Οι constructors της HeuristicPlayer υλοποιούνται με χρήση της super( ), που εν προκειμένω αφορά την κλάση Player. Τέλος, οι getters και setters των μεταβλητών έχουν πολύ απλή υλοποίηση.

Αποφασίσαμε την προσθήκη επιπλέον συνάρτησης, της seeAround( ), διότι χρειάζεται, σε άλλες συναρτήσεις, ο παίκτης να παρατηρήσει τα πλακίδια γύρω του και η επανάληψη ενός μακροσκελούς τμήματος κώδικα υποβαθμίζει τη αναγνωσιμότητα. Για το λόγο, λοιπόν, ότι ο ρόλος της είναι επικουρικός, αναλύεται πρώτη.

1.1. Συνάρτηση seeAround( )

Η συνάρτηση αυτή επιστρέφει ένα πίνακα int δύο στοιχείων, που περιέχει την απόσταση του παίκτη από κάποιο λάφυρο και από τον Μινώταυρο, εφόσον βρίσκονται στα ορατά πλακίδια της οριζόμενης κατεύθυνσης.

- **Ορισμός - Αρχικοποίηση Μεταβλητών**

Η εν λόγω συνάρτηση λαμβάνει ως ορίσματα την τρέχουσα θέση του παίκτη, τη θέση του αντιπάλου και την κίνηση (ζάρι) την οποία ενδιαφερόμαστε να πραγματοποιήσουμε. Την κατεύθυνση, δηλαδή, για την οποία επιδιώκουμε να συλλέξουμε πληροφορίες.

Ορίζονται οι μεταβλητές τύπου int blocksToOpponent και blockToSupply, οι οποίες, όπως προδίδουν τα ονόματά τους, είναι ενδεικτικές των αποστάσεων από τον αντίπαλο και το κοντινότερο λάφυρο, αντίστοιχα. Αρχικοποιούνται στη μέγιστη τιμή των μεταβλητών int, συμβολίζοντας ότι δεν έχουν εντοπιστεί τα ζητούμενα.

- **Το κύριο σώμα της συνάρτησης**

Όπως διαπιστώσαμε στο Α' μέρος της εργασίας, η αναζήτηση προς κάποια κατεύθυνση σε έναν πίνακα χαρακτηρίζεται από πολλές περιπτώσεις και ελέγχους. Συνεπώς, επιλέξαμε τη χρήση switch για την καλύτερη ομαδοποίηση των ελέγχων.

Ας λάβουμε την περίπτωση της κατεύθυνσης προς τα άνω (ζάρι με τιμή 1). Καταρχάς, επιβάλλεται να ελέγξουμε αν υπάρχει τοίχος στο πλακίδιο του παίκτη που εμποδίζει την ορατότητα:

```
if(board.getTiles()[currentPos].getUp()) {
	break;
}
```

Εφόσον βεβαιωθούμε ότι δεν υπάρχει τοίχος, μπορούμε να ελέγξουμε εάν ο δείκτης του αμέσως ανώτερου πλακιδίου ταυτίζεται με αυτόν του αντιπάλου:

```
if(opponentPos == currentPos + board.getN()) {
	blocksToOpponent = 1;
}
```

καθώς, ως γνωστόν, εάν στο δείκτη στοιχείου πίνακα N * N προσθέσουμε N, παίρνουμε το δείκτη του ανώτερου από το αρχικό στοιχείου.

Για τον εντοπισμό λαφύρου στο ελεγχθέν πλακίδιο χρειάζεται βρόγχος for, προκειμένου να επαληθευθεί εάν υπάρχει supply με supplyTileId ίδιο με το id του πλακιδίου. Ωστόσο, απαιτούνται περαιτέρω συνθήκες:

- Το λάφυρο πρέπει να είναι διαθέσιμο, δηλαδή η μεταβλητή του obtainable να έχει την τιμή true.
- Εφόσον αναζητούμε το κοντινότερο ορατό λάφυρο, η τοπική μεταβλητή blocksToSupply χρειάζεται να έχει την default τιμή της, Integer.MAX_VALUE. Σε περίπτωση που παραλείπονταν αυτός ο έλεγχος, και υπήρχαν δύο διαδοχικά λάφυρα σε μια στήλη, η μεταβλητή αυτή θα έπαιρνε την τιμή του ανώτατου, αγνοώντας ό,τι παρεμβάλλεται.

Όπως προκύπτει από τη λογική που ακολουθούμε, ο περιγραφόμενος κώδικας εμφωλεύεται σε επαναληπτικό βρόγχο for ώστε να ελέγχονται ability (το πολύ) σε πλήθος πλακίδια. Τέλος, στην περίπτωση που εντοπιστεί και λάφυρο και ο αντίπαλος πριν τελειώσει ο βρόγχος, επιθυμούμε να διακοπεί πρόωρα, καθώς έχουμε συλλέξει τις απαιτούμενες πληροφορίες. Άρα προστίθεται ο έλεγχος:

```
if(blocksToOpponent != Integer.MAXVALUE && blocksToSupply != Integer.MAXVALUE){
	break;
}
```

ο οποίος ελέγχει εάν οι δύο μεταβλητές έχουν ακόμα την προκαθορισμένη τιμή τους.

Ανάλογα υλοποιούνται και οι υπόλοιπες κατευθύνσεις.

Εν τέλει, επιστρέφεται ο πίνακας:

```
int[] tempArray = {blocksToSupply, blocksToOpponent};
```

1.2. Συνάρτηση evaluate( )

Αποτελεί την βασική συνάρτηση λογικής του έξυπνου παίκτη. Θα περιμέναμε να είναι και η εκτενέστερη συνάρτηση, ωστόσο εδώ χρησιμεύει η seeAround( ), αφού μεγάλο τμήμα «διαδικαστικού» κώδικα εκτελείται μέσω αυτής.

- **Ορισμός - Αρχικοποίηση Μεταβλητών**

Εκτός από τα ορίσματα που ζητήθηκαν, δηλαδή την τρέχουσα θέση του παίκτη και την πιθανή ζαριά του, προστέθηκε, για λόγους ευκολίας και η θέση του αντιπάλου. Αν και φαινομενικά αυτό παραβαίνει τους κανόνες του παιχνιδιού αλλά είναι απαραίτητη για την seeAround( ).

Αρχικοποιείται, λοιπόν, ο πίνακας (int[ ]) observation με αυτόν που επιστρέφει η seeAround(). Οι δύο τιμές του πίνακα αποθηκεύονται στις τοπικές μεταβλητές blocksToSupply και blocksToOpponent, για λόγους σαφήνειας.

- **Το κύριο σώμα της συνάρτησης**

Ως συνάρτηση αξιολόγησης επιλέχθηκε η παράσταση:

```
0.5/(blocksToSupply - 1) - 1.0/(blocksToOpponent - 1),
```

Ας αναλύσουμε βαθμηδόν την παράσταση.

Οι αποστάσεις είναι στους παρονομαστές.

Οι μεταβλητές στους παρονομαστές εξασφαλίζουν την αύξηση της επιστρεφόμενης τιμής με την ελάττωση της απόστασης του παίκτη από την αντίστοιχη μεταβλητή. Αυτό έχει ως αποτέλεσμα να αξιολογούνται καλύτερα κινήσεις που οδηγούν σε λάφυρα από ότι στον Μινώταυρο.

Το «-1» στον παρονομαστή.

Οι μεταβλητές ελαττώνονται κατά ένα, έτσι ώστε να έχουμε μέγιστη τιμή αξιολόγησης (+∞) όταν η απόσταση από λάφυρο είναι ένα, και ελάχιστη τιμή (-∞) εάν η απόσταση από τον Μινώταυρο είναι ένα. Η επιλογή αυτή βασίζεται στο γεγονός ότι

τέτοιες κινήσεις κρίνουν άμεσα την έκβαση του παιχνιδιού. Η περίπτωση όπου και οι δύο αποστάσεις είναι ένα καλύπτεται αργότερα.

Οι αριθμητές των κλασμάτων διαφέρουν.

Η απόσταση από ένα λάφυρο οφείλει να έχει μικρότερη επίδραση στο αποτέλεσμα, αφού η επιβίωση του παίκτη υπερτερεί της συγκέντρωσης λαφύρων. Για αυτό ο αριθμητής του πρώτου κλάσματος είναι μικρότερος.

Η λειτουργία της συνάρτησης αξιολόγησης

Η αξιολόγηση μιας κίνησης που δεν έχει κάποια ορατά στοιχεία, δηλαδή το αποτέλεσμα της seeAround( ) είναι πίνακας με τιμές:

```
{Integer.MAX_VALUE, Integer.MAX_VALUE},
```

οπότε προκύπτει αξιολόγηση πολύ μικρής απόλυτης τιμής.

Η συνάρτηση λειτουργεί όπως αναμένεται για την πλειοψηφία των περιπτώσεων, ωστόσο εάν οι δύο μεταβλητές έχουν τιμή ένα, προκύπτει απροσδιοριστία (∞ - ∞). Όταν συμβαίνει αυτό, φυσικά, επιδιώκουμε ο Θησέας να μην προχωρήσει σε αυτή την κίνηση, εκτός κι αν πρόκειται για το τελευταίο λάφυρο του ταμπλό. Για αυτό το λόγο, προστίθεται if πριν από την επιστροφή:

```
if(blocksToOpponent == 1 && blocksToSupply == 1){
                if(score == board.getS() - 1){
                    return Double.POSITIVE_INFINITY;
                }
                return Double.NEGATIVE_INFINITY;
}
```

Έπειτα, συναντούμε την περίπτωση:

		S1	T	S2	M		

όπου T ο Θησέας, M ο Μινώταυρος και S1 και S2 δύο τυχαία λάφυρα. Σύμφωνα με τη συνάρτηση, οι αξιολογήσεις των κινήσεων δεξιά και αριστερά είναι άπειρο. Ωστόσο, η κίνηση δεξιά ενέχει κινδύνους, αφού ο Μινώταυρος έχει αυξημένη πιθανότητα να πιάσει το Θησέα. Το ίδιο ισχύει εάν το S1 λάφυρο βρίσκεται σε οποιαδήποτε από τις εικονιζόμενες αριστερές του Θησέα θέσεις. Η κίνηση δεξιά θα ήταν επιθυμητή μόνο εάν το λάφυρο S2 ήταν το τελευταίο στο ταμπλό. Αυτό αποτυπώνεται στον κώδικα παρακάτω.

```
if(score == board.getS() - 1 && blocksToSupply == 1 && blocksToOpponent == 2)
```

```
return Double.POSITIVE_INFINITY;
```

Η συνάρτηση που συντάσσουμε επιθυμούμε να περιλαμβάνει και την περίπτωση που ο παίκτης είναι ο Μινώταυρος, επομένως η συνάρτηση τροποποιείται ως εξής:

```
0.5/(blocksToSupply) + 1.0/(blocksToOpponent - 1)
```

Παρατηρείστε ότι η παράσταση απειρίζει μόνο όταν ο Μινώταυρος είναι δίπλα στο Θησέα. Επίσης, ο Μινώταυρος επιδιώκει και αυτός να κινείται προς τα λάφυρα με σκοπό να στήσει ενέδρα για τον Θησέα.

1.3. Συνάρτηση getNextMove( )

Η συνάρτηση αξιολόγησης evaluate( ) επιστρέφει τη «βαθμολογία» μιας ζαριάς, οπότε για να βρεθεί η κατάλληλη κίνηση απαιτείται σύγκριση και των τεσσάρων υποψήφιων κινήσεων. Τον ρόλο αυτόν αναλαμβάνει η getNextMove(), η οποία επιστρέφει την καταλληλότερη ζαριά και ανανεώνει το ιστορικό των κινήσεων που αποθηκεύεται στο δυναμικό πίνακα path.

- **Ορισμός - Αρχικοποίηση Μεταβλητών**

Η συνάρτηση δέχεται ως ορίσματα τις τρέχουσες θέσεις του παίκτη και του αντιπάλου του. Στην αρχή, ορίζουμε τον πίνακα τύπου double[ ] movesValues, ο οποίος θα περιέχει τις αξιολογήσεις όλων των πιθανών ζαριών, δεικτοδοτούμενος με ωρολογιακή φορά ξεκινώντας από την κίνηση προς τα επάνω. Αρχικοποιούνται και οι απαραίτητες μεταβλητές για την εύρεση της ζαριάς με την υψηλότερη βαθμολογία.

- **Το κύριο σώμα της συνάρτησης**

Ο εντοπισμός της καταλληλότερης ζαριάς πραγματοποιείται με επαναληπτικό βρόγχο for. Εντός του βρόγχου, συγκρίνεται κάθε αξιολόγηση με τη μέγιστη αξιολόγηση που έχει βρεθεί μέχρι στιγμής (μεταβλητή double maxValue) και αν η εκάστοτε αξιολόγηση είναι μεγαλύτερη, καθίσταται αυτή η μέγιστη τιμή. Αποθηκεύεται και η αντίστοιχη ζαριά στη μεταβλητή (int) maxValueDie. Σε περίπτωση ισοβαθμίας όλων των ζαριών, δεν θα ήταν επιθυμητό ο Θησέας να επιλέγει πάντα την ίδια κίνηση διότι αυτό θα μπορούσε να οδηγήσει σε επανειλημμένη απώλεια της σειράς του. Για αυτό λαμβάνονται οι τυχαία αρχικοποιημένες τιμές των δύο αυτών μεταβλητών, όπως φαίνεται στον κώδικα πριν τον βρόγχο.

Στη συνέχεια, για να ανανεωθεί ο δυναμικός πίνακας path με την καινούρια κίνηση, ορίζεται πίνακας (Integer[ ]) tempArray, του οποίου οι τιμές έχουν ως εξής:

```
{maxValueDie, 0, observation[0] - 1, observation[1] - 1}
```

όπου maxValueDie η τιμή της προαναφερθείσας μεταβλητής, observation[0] και observation[1] οι τιμές του πίνακα που επιστρέφεται από την evaluate( ). Ο αριθμός 0 συμβολίζει στο δεύτερο στοιχείο του πίνακα δηλώνει ότι ο Θησέας δεν έχει συλλέξει λάφυρο. Ωστόσο, τέτοιος έλεγχος δεν έχει πραγματοποιηθεί και για αυτό λόγω προστίθεται:

```
if(name.equals("Theseus") && maxValue == Double.POSITIVE_INFINITY)
      tempArray[1] = 1;
```

Καθώς, αφενός, μόνο ο Θησέας συλλέγει λάφυρα και αφετέρου η μόνη περίπτωση η μεταβλητή maxValue να λάβει την τιμή (+∞) είναι όταν η απόστασή του από κάποιο λάφυρο στην κατεύθυνση κίνησής του είναι ένα. Άρα προφανώς το συλλέγει.

Τέλος, μέσω της συνάρτησης add( ) προσθέτουμε τον πίνακα tempArray στο path και επιστρέφεται η πιο υψηλόβαθμη ζαριά.

1.4. Συνάρτηση statistics( )

Αποτελεί τη συνάρτηση που εκτυπώνει όλες τις κινήσεις που εκτελεί ο παίκτης σε ένα συγκεκριμένο παιχνίδι, καθώς και άλλες πληροφορίες σχετικά με τα λάφυρα. Χρησιμεύει στο να μελετήσουμε τις κινήσεις του παίκτη στο τέλος του παιχνιδιού.

- **Ορισμός - Αρχικοποίηση Μεταβλητών**

Προκειμένου να αθροιστούν όλες οι κινήσεις των επιμέρους κινήσεων ορίζονται οι αντίστοιχες μεταβλητές (int) ups, right, downs και lefts και αρχικοποιούνται στο 0.

- **Το κύριο σώμα της συνάρτησης**

Επιθυμούμε στην αρχή να εκτυπώνονται οι κινήσεις που επιλέχθηκαν από τη συνάρτηση getNextMove(). Έτσι, αξιοποιώντας τον πίνακα path, λαμβάνουμε τον αριθμό της ζαριάς στον κάθε γύρο και με χρήση switch εκτυπώνεται το αντίστοιχο μήνυμα μετακίνησης. Παράλληλα, αυξάνεται και ο αντίστοιχος μετρητής κινήσεων προς την αντίστοιχη κατεύθυνση. Με ανάλογο τρόπο, εκτυπώνεται μήνυμα για την επιτυχία συλλογής λαφύρου, την απόστασή του από το κοντινότερο ή την αδυναμία εύρεσής του.

Η παραπάνω διαδικασία εμφωλεύεται σε επαναληπτικό βρόγχο που πραγματοποιείται τόσες φορές όσες και οι ζαριές του παίκτη, δηλαδή όσες το μέγεθος του πίνακα path. Τελικά, αφού τελειώσει ο βρόγχος, οι μετρητές των κινήσεων περιέχουν το σύνολο των αντίστοιχων κινήσεων σε ολόκληρο το παιχνίδι και εκτυπώνονται με αντίστοιχη εντολή.

1.5. Συνάρτηση move( )

Η συνάρτηση αυτή αναλαμβάνει τη μετακίνηση του παίκτη με δεδομένο αριθμό ζαριού. Είναι πανομοιότυπη με την move( ) της κλάσης Player που αναλύθηκε στο Α΄ μέρος της εργασίας, με μία μόνο εξαίρεση: η κίνηση δεν επιλέγεται τυχαία, αλλά αποτελεί όρισμα της συνάρτησης, προκειμένου να πραγματοποιεί τις έξυπνες κινήσεις που προκύπτουν χάρη στις προηγούμενες συναρτήσεις. Επομένως, δεν υπάρχει κάτι αξιοσημείωτο προς αναφορά.

2. Κλάση Game

Η κλάση Game παρέμεινε σταθερή όσον αφορά τα instance fields και τις συναρτήσεις της, εκτός της main, η οποία δέχτηκε μικρές τροποποιήσεις, προκειμένου να λειτουργεί έχοντας ως παίκτες αντικείμενα της κλάσης HeuristicPlayer, και όχι της Player.

2.1. Συνάρτηση main( )

- **Ορισμός - Αρχικοποίηση Μεταβλητών**

Η main ξεκινάει με τη δημιουργία ενός παιχνιδιού, δηλαδή ενός αντικειμένου Game, καθώς και την αρχικοποίηση των παραμέτρων που θα χρειαστούν για το αντικείμενο Board (διάσταση ταμπλό, αριθμός εφοδίων, αριθμός τοίχων και αριθμός γύρων). Μετά την αρχικοποίηση του board, καλείται η συνάρτηση που το μετατρέπει σε τυχαίο λαβύρινθο, η createBoard( ). Δημιουργούνται και οι έξυπνοι παίκτες, αντικείμενα της κλάσης HeuristicPlayer με τα οριζόμενα στοιχεία. Ορίζεται και μια βοηθητική μεταβλητή, η (int) winnerIdx, που υποδηλώνει το id του νικητή, και αρχικοποιείται στο -1, για λόγο που θα φανεί παρακάτω.

- **Το κύριο σώμα της συνάρτησης**

Για την εξέλιξη του παιχνιδιού μέσω της αύξησης του αριθμού του γύρου, επιλέχθηκε επαναληπτικός βρόγχος do-while για εμφανείς λόγους. Ο κώδικας στο βρόγχο δεν αποκλίνει σημαντικά από αυτόν στο Α΄ μέρος.

Μόλις τελειώσει ή, σε περίπτωση νίκης κάποιου παίκτη, διακοπεί ο βρόγχος, εκτυπώνονται τα στατιστικά του Θησέα με χρήση της statistics( ). Εκτυπώνεται, στη συνέχεια, μήνυμα που ανακοινώνει ισοπαλία ή νικητή. Εδώ καθίσταται προφανής η λειτουργία της μεταβλητής winnerIdx· λαμβάνει τιμή ίση με το id του παίκτη μόνο όταν εκείνος κερδίσει το παιχνίδι. Συνεπώς, αν το παιχνίδι κριθεί νικηφόρο για κάποιον παίκτη στον τελευταίο γύρο, δεν εμφανίζεται το μήνυμα της ισοπαλίας.

Και κάπως έτσι το παιχνίδι φτάνει στο τέλος του...