Θησέας και Μινώταυρος

Εργασία στις Δομές Δεδομένων, Μέρος Β

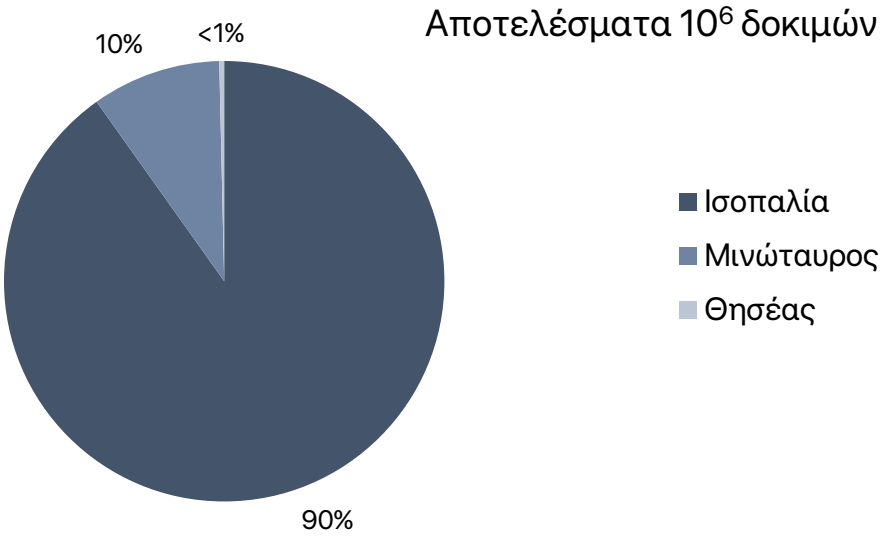
Τερζίδης Αλέξανδρος

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, αποφεύγοντας ταυτόχρονα το 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 παρατηρήσει γειτονικά του πλακίδια και να αξιολογεί την κάθε του κίνηση. Είναι λογικό να επιδιώκει να κινείται προς τα εφόδια 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 που δύναται να λειτουργήσει βάσει κάποιων κριτηρίων (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 βοηθητικές μεταβλητές (int) ability και (double) chanceToFindTreasure. Η πρώτη καθορίζει το μήκος (σε πλήθος tiles) της «όρασης» παίκτη και η δεύτερη ******. 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, 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) για 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 της απόστασης 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 εφοδίου 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 εφόδιο, 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 εφόδια σε μια στήλη, η μεταβλητή αυτή θα έπαιρνε την τιμή του πιο ανώτερου, αγνοώντας αυτό που 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. Αυτό εξασφαλίζεται προσθέτοντας i * N ή (i + 1) * N στο δείκτη του πλακιδίου, επιλέγοντας το δεύτερο όταν θέλουμε να αποσπάσουμε πληροφορίες για το αμέσως ανώτερο πλακίδιο. Τέλος, στην περίπτωση που εντοπιστεί και λάφυρο και ο αντίπαλος πριν τελειώσει ο βρόγχος, επιθυμούμε να διακοπεί πρόωρα, άρα προστίθεται σε αυτόν και ο έλεγχος:

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

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

Με μικρές τροποποιήσεις όσον αφορά τους δείκτες των πλακιδίων 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, στην πραγματικότητα δεν ισχύει, αφού, όπως εξηγήθηκε, η συνάρτηση που χρησιμοποιείται για την ανάλυση των απαραίτητων πλακιδίων, εάν δεν εντοπίσει τον αντίπαλο, επιστρέφει την default τιμή της.

Αρχικοποείται, λοιπόν, ο πίνακας (int[ ]) observation που λαμβάνει την τιμή που επιστρέφει η seeAround( ) με ορίσματα ίδια της εκάστοτε evaluate( ). Οι δύο τιμές του πίνακα αποθηκεύονται στις τοπικές μεταβλητές blocksToSupply και blocksToOpponent με τον προφανή τρόπο, για λόγους σαφήνειας.

- **Το κύριο σώμα της συνάρτησης**

Ως συνάρτηση αξιολόγησης επιλέχθηκε η παράσταση:

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

Επιλέχθηκε η συνάρτηση να περιέχει τα παραπάνω κλάσματα, καθώς:

- οι μεταβλητές στους παρονομαστές εξασφαλίζουν την αύξηση της επιστρεφόμενης τιμής με την ελάττωση της απόσταση του παίκτη από την αντίστοιχη μεταβλητή. Αυτό αποσκοπεί στη μεγαλύτερη αξιολόγηση κινήσεων που πλησιάζουν πλακίδια με λάφυρο και τη μικρότερη αυτών που πλησιάζουν το Μινώταυρο.
- οι μεταβλητές ελαττώνονται κατά ένα, έτσι ώστε να έχουμε μέγιστη τιμή αξιολόγησης (+∞) όταν η απόσταση από εφόδιο είναι ένα, και ελάχιστη τιμή

(-∞) εάν η απόσταση από τον Μινώταυρο είναι ένα. Η επιλογή αυτή βασίζεται στο γεγονός ότι τέτοιες κινήσεις, γενικά, χρειάζεται να υπερισχύουν ή όχι έναντι των τυχαίων. Η περίπτωση όπου και τα δύο είναι ένα καλύπτεται παρακάτω.

- το πρώτο κλάσμα οφείλει να έχει μικρότερη επίδραση στο αποτέλεσμα, αφού η επιβίωση του παίκτη υπερτερεί της συγκέντρωσης λαφύρων. Έτσι, ο αριθμητής του είναι μικρότερος του άλλου κλάσματος και τα δύο κλάσματα αφαιρούνται.
- 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} ,
```

Ο κώδικας που συντάσσουμε επιθυμούμε να καλύπτει και την περίπτωση που ο παίκτης είναι ο Μινώταυρος, επομένως για την αξιο

2. Κλάση Game