ΑΡΙΣΤΟΤΕΛΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ

Τμημα ΗΜΜΥ. Τομεας Ηλεκτρονικης

Δομες δεδομενων

Λαβύρινθος: Ο Θησέας και ο Μινώταυρος III

Ομαδα 182

Συντάκτες:
Αναστασια Φωτη
ΑΕΜ:8959

anastaskf@ece.auth.gr

Χρηστος Χουτουριδης
ΑΕΜ:8997

cchoutou@ece.auth.gr

Διδάσκων:
Σταυρουλα Σιαχαλου

ssiachal@auth.gr

20 Δεκεμβρίου 2020

1. Εισαγωγη

Η παρούσα εργασία αφορά την δημιουργία ενός παιχνιδιού με τίτλο "Μία νύχτα στο μουσείο". Στο συγκεκριμένο παιχνίδι καλούμαστε να υλοποιήσουμε ένα λαβύρινθο μέσα στον οποίο κινούνται δύο παίκτες, ο Μινώταυρος και ο Θησέας. Στόχος του Μινώταυρου είναι να "πιάσει" τον Θησέα και στόχος του Θησέα είναι να βρει όλα τα εφόδια πού βρίσκονται κατανεμημένα στον ταμπλό, πριν ξημερώσει και πριν τον "πιάσει" ο Μινώταυρος.

Υπενθυμίζουμε πως στις πρώτες εργασίες κληθήκαμε να υλοποιήσουμε το ταμπλό και τα στοιχεία που το απαρτίζουν καθώς και τις βασικές λειτουργίες του παιχνιδιού. Στη δεύτερη εργασία κληθήκαμε επιπρόσθετα να υλοποιήσουμε ένα παίκτη ο οποίος χειρίζεται το ζάρι προς όφελός του. Στην παρούσα εργασία ασχοληθήκαμε με τη δημιουργία ενός παίκτη που κάνει χρήση του αλγόριθμου minimax, με σκοπό να προβλέψει καλύτερα την κίνησή του, αναλύοντας τις πιθανές κινήσεις του αντιπάλου. Φυσικά όλα τα βασικά συστατικά του παιχνιδιού που αναπτύξαμε στις δύο πρώτες εργασίες είναι παρόντα. Οι τροποποιήσεις σε αυτά είναι ελάχιστες και αφορούν κυρίως την συμβατότητα με τις νέες λειτουργίες, όπως αυτές περιγράφονται στην εκφώνηση.

2. Παραδοτεα

Τα επισυναπτόμενα παραδοτέα αποτελούνται από:

- Τον **root** κατάλογο στον οποίο υπάρχει και το project του eclipse.
- Ένας υποκατάλογος **src/** με τον κώδικα της java, αποτελούμενο από ένα αριθμό αντικειμένων ενσωματωμένο στο πακέτο host.labyrinth.
- Ένας υποκατάλογος **out/** που περιέχει το παραγόμενο command line jar του παιχνιδιού.
- Ένας υποκατάλογος **doc/** με την τεκμηρίωση του κώδικα όπως αυτή έχει παραχθεί από τα σχόλια, με το εργαλείο doxygen. Το αρχείο ρυθμίσεων του doxygen είναι στον root με το όνομα Doxyfile. Η πλοήγηση στην τεκμηρίωση μπορεί να γίνει ανοίγοντας το αρχείο doc/html/index.html
- Το αρχείο **report.pdf** που αποτελεί την **παρούσα αναφορά**.

Εκτός από τα επισυναπτόμενα αρχεία διαθέσιμο υπάρχει και το **git αποθετήριο** ολόκληρης της εργασίας εδώ. Αυτό περιέχει τόσο τον κώδικα της εφαρμογής όσο και τον κώδικα της αναφοράς.

3. Σχεδιαστικες επιλογες

Πριν ασχοληθούμε όμως με τα ζητηθέντα αντικείμενα του προγράμματος, θα πρέπει να αναφερθούμε σε ορισμένες δομές που προστέθηκαν, αλλά και κάποιες σχεδιαστικές επιλογές που έγιναν για να απλοποιήσουν τον κώδικα. Αρκετές από αυτές προέρχονται από τις προηγούμενες εργασίες. Τις παραθέτουμε όμως και εδώ καθώς παίζουν σημαντικό ρόλο, τόσο στην λειτουργία του προγράμματος όσο και στην καλύτερη κατανόηση της τρέχουσας εργασίας.

3.1. Accessor - mutator idiom

Στις προδιαγραφές της εργασίας αφήνεται να εννοηθεί πως ζητείται η χρήση του *accessor - mutator idiom*. Θα πρέπει να παραδεχτούμε όμως, πως **θεωρούμε το συγκεκριμένο ιδίωμα ιδιαίτερα προβληματικό**. Ο κύριος λόγος είναι πως παραβιάζει θεμελιακά τις αφαιρέσεις προδίδοντας τον εσωτερικό σχεδιασμό του αντικειμένου. Ακόμα δίνει πρόσβαση στην εσωτερική δομή του, "πίσω από την πλάτη" του αντικειμένου. Εν αντιθέτως με το ιδίωμα αυτό, **τα αντικείμενα που υλοποιούνται ως αφαιρέσεις θα μπορούσαν να προσφέρουν μεθόδους που εκτελούν κάποια λειτουργία, κρύβοντας τελείως τις εσωτερικές λεπτομέρειες της υλοποίησης**. Αυτός είναι και ο δρόμος που διαλέξαμε για το σχεδιασμό του προγράμματος. Η κάθε τάξη του προγράμματός μας προσφέρει δημόσια ένα αριθμό από μεθόδους που είναι απαραίτητες για την εκάστοτε απαιτούμενη λειτουργικότητα και κρύβει όσο καλύτερα γίνεται την εσωτερική υλοποίηση. Ενώ λοιπόν υλοποιήσαμε το ζητηθέν get-set ζευγάρι για την κάθε μεταβλητή των τάξεων, δεν το χρησιμοποιήσαμε σχεδόν καθόλου μέσα στο πρόγραμμα.

3.2. Ενοποιημένο σύστημα συντεταγμένων

Στις προδιαγραφές της 1ης εργασίας περιγράφεται επίσης ένα διπλό σύστημα συντεταγμένων, τόσο για τα πλακίδια όσο και για τα εφόδια. Ένα καρτεσιανό που διευθυνσιοδοτεί ως προς δύο άξονες και περιέχει ένα ζευγάρι γραμμής και στήλης και ένα μονοδιάστατο που αποτελείται από τον γραμμικό συνδυασμό των προηγουμένων. Το μονοδιάστατο αντικατοπτρίζει και την απεικόνιση στη μνήμη ενός πίνακα 2 διαστάσεων σε row major order.

Γενικά θεωρούμε πως κάτι τέτοιο δημιουργεί πλεονασμό δεδομένων και επομένως είναι κακή πρακτική. Αυτό γιατί μεταξύ άλλων μειονεκτημάτων, που κυρίως αφορά τον πολυνηματικό προγραμματισμό, οδηγεί και σε προγραμματιστικά λάθη που πιθανώς θα αφήνουν τα δύο συστήματα ασυγχρόνιστα. Για να λύσουμε αυτό το πρόβλημα **δημιουργήσαμε την τάξη Position**, στην οποία εσωτερικά χρησιμοποιούμε μόνο το ένα από τα δύο συστήματα, για την ακρίβεια το μονοδιάστατο και ταυτόχρονα παρέχουμε μεθόδους για την πρόσβαση στη θέση και από τα δύο συστήματα. Η τάξη μεταξύ άλλων προσφέρει και **στατικές μεθόδους για τις μετατροπές** προσφέροντας έτσι μια είδους εργαλειοθήκη για την εφαρμογή. Για την παρούσα εργασία χρησιμοποιήσαμε την Position όπου ήταν δυνατό.

3.3. Αναβάθμιση της τάξης Tile

Κατά τον προγραμματισμό του παιχνιδιού παρατηρήσαμε πως τόσο η τάξη Tile όσο και η Supply έχουν πληροφορίες για τη θέση τους στο ταμπλό. Αυτό σημαίνει πως τόσο τα πλακίδια όσο και τα εφόδια ανήκουν στο ταμπλό. Ακόμα σημαίνει πως δημιουργούν επιπλέον πλεονασμό σε δεδομένα, καθώς απαιτείται οι συντεταγμένες του πλακιδίου που βρίσκεται το εφόδιο να επαναληφθούν μέσα στην Supply. Κάτι τέτοιο γίνεται αντιληπτό και από τις προδιαγραφές της Board η οποία είναι αυτή που περιέχει τους πίνακες αναφορών τόσο των πλακιδίων όσο και των εφοδίων. **Μια πιο διαισθητική προσέγγιση βέβαια θα ήθελε τα εφόδια να ανήκουν στα πλακίδια** και όχι στο ταμπλό. Με αυτό τον τρόπο η τάξη Supply δεν θα είχε τις επαναλαμβανόμενες πληροφορίες θέσης, αλλά αντίθετα η τάξη Tile θα είχε απλώς μια επιπρόσθετη πληροφορία για το αν φιλοξενεί κάποιο εφόδιο ή όχι.

Όπως είναι φυσικό θελήσαμε να υλοποιήσουμε αυτή την προσέγγιση. Αν όμως μετακινούσαμε τις αναφορές των εφοδίων στην Tile θα αλλοιώναμε τις προδιαγραφές της εκφώνησης. Επομένως επιλέξαμε μια μέση οδό. Εμπλουτίσαμε την Tile με μεθόδους που αφορούν τα εφόδια. Αυτές είναι οι:

- ***int hasSupply (Supply[] supplies)***
  που δίνει την δυνατότητα να ελέγξουμε αν ένα πλακίδιο έχει κάποιο ενεργό εφόδιο. Και
- ***void pickSupply (Supply[] supplies, int supplyId)***
  που δίνει την δυνατότητα σε κάποιο παίκτη να "σηκώσει" το εφόδιο.

Έτσι έχουμε ομοιομορφία με τις αντίστοιχες μεθόδους *boolean hasWall(int direction)* και *int hasWalls()* της Tile.

Ένας παρατηρητικός αναγνώστης θα διαπιστώσει πως οι συναρτήσεις για τα εφόδια είναι αναγκασμένες να πάρουν τον πίνακα αναφορών στα εφόδια ως όρισμα. Αυτό είναι το τίμημα που πρέπει να πληρώσουμε προωθώντας τις μεθόδους αυτές στην Tile.

3.4. Αναβάθμιση της τάξης Board

Στην διάρκεια του σχεδιασμού θεωρήσαμε ωφέλιμη την εισαγωγή του πίνακα *int[][] moves* στην τάξη Board. Σε αυτόν καταχωρούμε τις πληροφορίες της τελευταίας κίνησης του κάθε παίκτη, όπως το id του πλακιδίου που βρίσκεται ο παίκτης, οι συντεταγμένες του και η "ζαριά" της κίνησης. Στην προσθήκη αυτή οδηγηθήκαμε αφού παρατηρήσαμε πως για την τάξη HeuristicPlayer είναι απαραίτητη η πρόσβαση στην θέση και στο id του αντιπάλου. Φυσικά την ίδια απαίτηση έχουμε και με την τάξη MinMaxPlayer. Η πρόσβαση στις κινήσεις πραγματοποιείται μέσω της μεταβλητής moves και της συνάρτησης *getOpponentMove()*. Έτσι ο κάθε HeuristicPlayer ή MinMaxPlayer μπορεί να διαπιστώσει αν στο οπτικό του πεδίο βρίσκεται κάποιος αντίπαλος και να κινηθεί κατάλληλα.

Ακόμα μέσω του ίδιου μηχανισμού δίνουμε την δυνατότητα στους παίκτες να μαθαίνουν το ID του αντιπάλου. Αυτό είναι χρήσιμο για τον MinMaxPlayer όταν προσπαθεί να σκηνοθετήσει τις κινήσεις του αντιπάλου. Θα αναλύσουμε περισσότερα όμως γύρω από αυτό στην παράγραφο 5.1.

3.5. Αναβάθμιση της τάξης Player

Για την επέμβαση σε αυτή την τάξη κινηθήκαμε με γνώμονα την αντιμετώπιση δύο προβλημάτων που μας παρουσιάστηκαν. Αρχικά για την σωστή λειτουργία τη κλάσης HeuristicPlayer, όπως αναφέραμε και παραπάνω, είναι αναγκαία η πρόσβαση στο id του αντιπάλου. Για τον λόγο αυτό επιλέξαμε να δημιουργείται ένα id για κάθε παίκτη μέσω συνάρτησης της τάξης Board, στο οποίο θα έχουμε πρόσβαση μέσω του πίνακα *int[][] moves*. Τη λειτουργία αυτή αναλαμβάνει η συνάρτηση *int generatePlayerId()*.

Επιπρόσθετα, θελήσαμε να ενοποιήσουμε τις υλοποιήσεις των τάξεων Player, HeuristicPlayer και MinMaxPlayer, ώστε να τις κάνουμε συμβατές με την αρχή του Liskov [1]. Για το λόγο αυτό οι συναρτήσεις *move(), statistics()* και *final_statistice()* είναι πλέον υλοποιημένες και σε όλα τα αντικείμενα. Αυτό μας ανάγκασε να μετακινήσουμε την μεταβλητή path στο base object δηλαδή στην Player.

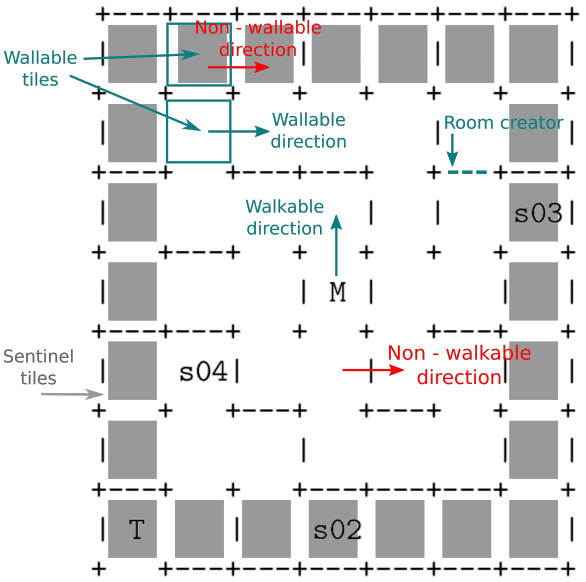
4. Concepts

Αν και τα concepts υλοποιήθηκαν για την εκπόνηση του πρώτου μέρους, τα παραθέτουμε συνοπτικά και σε αυτή την αναφορά, καθώς τα θεωρούμε ουσιαστικό κομμάτι για όλα τα μέρη της εργασίας. Τα "concepts" μας έχουν τη μορφή συναρτήσεων και αφορούν έννοιες σχετικές με την εφαρμογή. Μας δίνεται έτσι η δυνατότητα να ελέγχουμε αν κάποια είσοδός ενός predicate-concept πληροί τις προδιαγραφές του ή όχι. Στο σχήμα 1 φαίνεται μια οπτικοποιημένη έκδοση τους.

4.1. Πλακίδιο φρουρός - *isSentinel()*

Πρόκειται για concept που μας επιτρέπει να ελέγξουμε αν το πλακίδιο είναι *"πλακίδιο φρουρός"*. Αν δηλαδή βρίσκεται στα εξωτερικά άκρα του ταμπλό. Η υλοποίηση αυτού του concept γίνεται μέσω τεσσάρων συναρτήσεων που ελέγχουν χωριστά τις τέσσερεις διευθύνσεις του ταμπλό.

Για την παράδειγμα η *boolean isLeftSentinel (int tileId)* μας δίνει την δυνατότητα να ελέγξουμε αν το πλακίδιο είναι *πλακίδιο φρουρού* αριστερά του ταμπλό. Αυτό είναι χρήσιμο για λειτουργίες όπως για παράδειγμα αν θέλουμε να δούμε μήπως χρειάζεται να τοποθετηθεί εξωτερικός τοίχος αριστερά του πλακιδίου. Αντίστοιχα υπάρχουν και οι υπόλοιπες συναρτήσεις για τις υπόλοιπες διευθύνσεις.



Σχήμα 1: *Οπτική αναπαράσταση των concepts που χρησιμοποιούμε σε ένα ταμπλό 7x7. Με πράσινο αναπαρίστανται όσα πληρούν κάποιο concept και με κόκκινο όσα όχι. Τα πλακίδια με το γκρι χρώμα, είναι τα πλακίδια φρουροί.*

4.2. Διασχίσιμη διεύθυνση - *isWalkable()*

Πρόκειται για predicate που μας επιτρέπει να ελέγξουμε αν μια διεύθυνση σε κάποιο πλακίδιο είναι *"διασχίσημη διεύθυνση"*. Αν δηλαδή κάποιος παίκτης μπορεί να κινηθεί σε αυτή. Για να ισχύει κάτι τέτοιο θα πρέπει:

- Η διεύθυνση να μην έχει τοίχο.
- Η διεύθυνση να μην είναι η "Κάτω" διεύθυνση του πλακιδίου εισόδου στο λαβύρινθο.

[1] https://en.wikipedia.org/wiki/Liskov_substitution_principle

4.3. Χτίσιμη διεύθυνση - *isWallableDir()*

Πρόκειται για predicate που μας επιτρέπει να ελέγξουμε αν μια διεύθυνση κάποιου πλακιδίου είναι *"χτίσιμη διεύθυνση"*. Αν δηλαδή μπορούμε να τοποθετήσουμε τοίχο στη διεύθυνση αυτή. Για να είναι μια διεύθυνση χτίσιμη θα πρέπει:

- Η διεύθυνση να είναι *διασχίσιμη*.
- Το γειτονικό πλακίδιο σε αυτή τη διεύθυνση να μην περιέχει ήδη τον μέγιστο επιτρεπτό αριθμό τοίχων.
- Ο τοίχος να μην δημιουργεί κάποιο κλειστό δωμάτιο.

Αυτή η τελευταία **απαίτηση δεν υπάρχει στις προδιαγραφές** και την παίρνουμε υπόψιν, όπως θα δούμε και στην ενότητα 6, μόνο αν ο χρήστης την έχει ζητήσει από την γραμμή εντολών. Ο λόγος είναι γιατί ο υπολογισμός της κοστίζει και αυτό μπορεί να μην παίζει ρόλο για ταμπλό μεγέθους $15x15$, αλλά αν ζητηθεί κάποιο πολύ μεγαλύτερο τότε ο χρόνος είναι υπολογίσιμος. Φυσικά στον κώδικα κάνουμε χρήση αυτού του concept μόνο κατά τη δημιουργία του ταμπλό, με αποτέλεσμα να μην επιβαρύνεται καθόλου η λειτουργία του προγράμματος κατά τη διάρκεια του παιχνιδιού. Έτσι προτείνουμε στον αναγνώστη και χρήστη της εφαρμογής μας να κάνει χρήση αυτής της επιλογής. Αναφερόμαστε αναλυτικά σε αυτό τον αλγόριθμο στην ενότητα 4.5

4.4. Χτίσιμο πλακίδιο - *isWallable()*

Πρόκειται για predicate που μας επιτρέπει να ελέγξουμε αν κάποιο πλακίδιο είναι *"χτίσιμο πλακίδιο"*. Αν δηλαδή υπάρχει κάποια πλευρά του πλακιδίου στην οποία μπορούμε να τοποθετήσουμε τοίχο. Για να ισχύει αυτό θα πρέπει:

- Να υπάρχει τουλάχιστον μία *χτίσιμη διεύθυνση* στο πλακίδιο.
- Το πλακίδιο να μην έχει ήδη τον μέγιστο επιτρεπτό αριθμό τοίχων.

4.5. Δημιουργός κλειστού δωματίου - *isRoomCreator()*

Οι προδιαγραφές της εργασίας που αφορούν τη δημιουργία του ταμπλό, δεν αποτρέπουν την εμφάνιση κλειστών δωματίων. Για το λόγο αυτό υλοποιήσαμε ένα αλγόριθμο που ανιχνεύει την πιθανότητα δημιουργίας κλειστών δωματίων του οποίου η ενεργοποίηση γίνεται κατόπιν επιλογής του χρήστη από τη γραμμή εντολών.

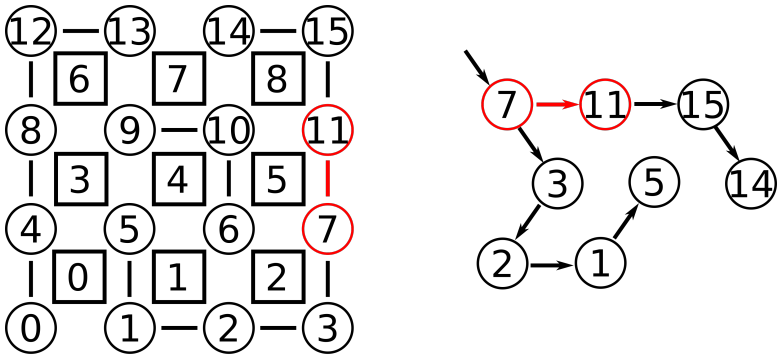
Ένας αλγόριθμος για να είναι λειτουργικός χρειάζεται δεδομένα. Στη δική μας περίπτωση τα δεδομένα είναι οι τοίχοι. Θέλαμε λοιπόν ένα τρόπο αναπαράστασης των τοίχων που να βολεύει για το συγκεκριμένο πρόβλημα. Η λύση που χρησιμοποιήσαμε συνοψίζεται στα εξής:

- Οι γωνίες των πλακιδίων ονομάζονται κόμβοι και η θέση τους αναπαρίστανται με έναν αύξοντα αριθμό που ισοδυναμεί με την αναπαράσταση δισδιάστατου πίνακα σε row major order.
- Ο κάθε κόμβος αποτελεί τον κόμβο-γωνία (vertex) ενός γράφου.
- Ο κάθε τοίχος αναπαρίσταται ως ακμή (edge) στον γράφο.

Για παράδειγμα στο σχήμα 2, που φαίνεται και ένα παράδειγμα αναπαράστασης, ο τοίχος αριστερά του πλακιδίου `3` αναπαρίσταται ως η ακμή `(4,8)`.

Για τον αλγόριθμο υλοποιήσαμε δύο τάξεις. Την **Edge** που δημιουργεί ζευγάρια κόμβων ώστε να μπορεί να αποθηκεύσει τον κάθε τοίχο(ακμή) και την **Graph** που προσφέρει λειτουργίες δημιουργίας συνεκτικού γράφου λαμβάνοντας ως είσοδο τοίχους(ακμές).

Η λειτουργία του είναι απλή. Κάθε φορά που ελέγχουμε αν μία διεύθυνση πλακιδίου είναι *χτίσιμη*, **δημιουργούμε το μεγαλύτερο δυνατό συνεκτικό γράφο που περιέχει τον εν λόγο τοίχο** και όλους τους ήδη τοποθετημένους τοίχους. Αν στον γράφο που προκύπτει υπάρχει κάποιος κόμβος περισσότερες από μία φορές, δηλαδή ο γράφος δεν είναι απλός, τότε αυτό σημαίνει πως στον εν λόγο κόμβο μπορούμε να πάμε ακολουθώντας τοίχους από τουλάχιστον δύο κατευθύνσεις. Άρα το



Σχήμα 2:
(α) Διευθυνσιοδότηση πλακιδίων(τετραγωνάκια) και κόμβων(σφαίρες) ενός ταμπλό 3x3.
(β) Ένας συνεκτικός γράφος που προκύπτει από το (α) ξεκινώντας από τον τοίχο (7, 11). Για τον εν λόγο τοίχο θέλουμε να διαπιστώσουμε αν δημιουργεί κλειστό δωμάτιο. Κάτι που σε αυτή την περίπτωση δεν συμβαίνει.

ταμπλό περιέχει κάποιο κλειστό δωμάτιο. Αφού τον αλγόριθμο τον εκτελούμε για κάθε πιθανή *χτίσιμη διεύθυνση*, τότε ο τοίχος που προκαλεί το κλειστό δωμάτιο είναι ο τρέχον.

Για την λειτουργία του αλγόριθμου χρειαζόμαστε όλους τους τοίχους που είναι ήδη τοποθετημένοι στο ταμπλό στη μορφή Edge. Γιαυτό προσθέσαμε στην τάξη Board μια λίστα αναφορών (ArrayList) και σε αυτήν αποθηκεύουμε κάθε τοίχο που δημιουργούμε. Τον αλγόριθμο μπορούμε να τον ενεργοποιήσουμε αν περάσουμε στο πρόγραμμα ως επιλογή το όρισμα `--norooms` από τη γραμμή εντολών [2].

Δυστυχώς η κωδικοποίηση που χρησιμοποιούμε εδώ δεν ταιριάζει με αυτή της υπόλοιπης εφαρμογής. Αυτό έχει σαν αποτέλεσμα να πρέπει να δημιουργούμε τον γράφο κάθε φορά. Αυτό έχει κόστος σε χρόνο. Για την ακρίβεια $O(N^2 logN)$, όπου N, ο συνολικός αριθμός πλακιδίων του ταμπλό. Αυτό το κόστος αφορά τον έλεγχο του κάθε πλακιδίου. Για όλο το ταμπλό το κόστος είναι $O(N^4 logN)$. Φυσικά θα μπορούσαμε να χρησιμοποιήσουμε υπομνηματισμό και να αποθηκεύουμε τους γράφους Όμως λόγο του ότι η επιβάρυνση λαμβάνει χώρα μόνο μία φορά κατά την εκκίνηση, σε συνδυασμό με το μικρό μέγεθος του ταμπλό, αποφασίσαμε να μην προχωρήσουμε σε περαιτέρω βελτιστοποίηση.

5. Υλοποιηση

Για την μεταγλώττιση της εφαρμογής, απαιτείται java έκδοση 8 ή και μεταγενέστερη καθώς έχουμε κάνει χρήση lambdas. Για υπενθύμιση, αναφέρουμε συνοπτικά τη λειτουργία ορισμένων αντικειμένων που προσθέσαμε κατά την πρώτη και δεύτερη εργασία.

- **Const**
  Το αντικείμενο αυτό περιέχει σταθερές για όλη την εφαρμογή.
- **Session**
  Το αντικείμενο αυτό περιέχει όλες τις τιμές της εφαρμογής που αποτελούν ρυθμίσεις ή επιλογές, όπως πχ το μέγεθος του ταμπλό, τον αριθμό των εφοδίων κτλ.
- **Direction**
  Το αντικείμενο αυτό λειτουργεί σαν C++ enumerator και παρέχει ονοματολογία στις διευθύνσεις που χρησιμοποιούμε στην εφαρμογή.
- **DirRange**
  Ομοίως ένα βοηθητικό αντικείμενο αυτή τη φορά για την αυτόματη δημιουργία διευθύνσεων σε βρόχους επανάληψης.
- **Edge**
  Το αντικείμενο αυτό όπως είδαμε και στην ενότητα 4.5, λειτουργεί ως αναπαράσταση των τοίχων με τη μορφή ακμών ενός γράφου. Προσφέρει constructor που δέχεται για ορίσματα συντεταγμένες από

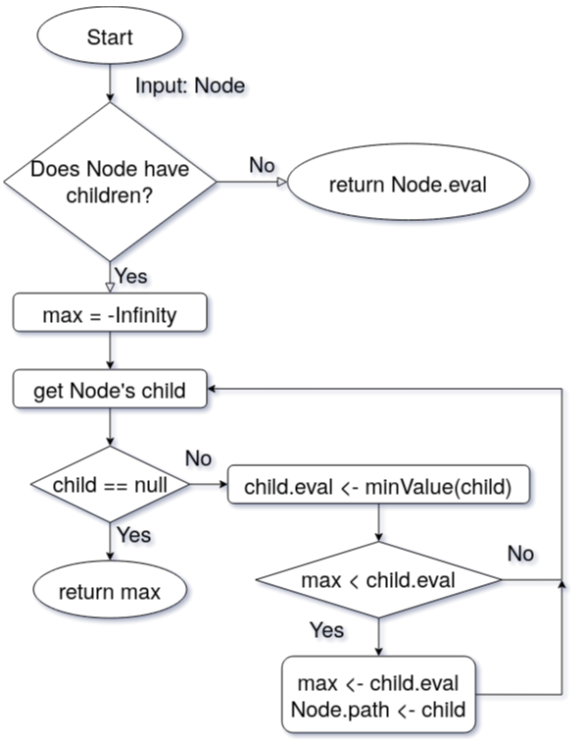
[2] Συνίσταται ανεπιφύλακτα η χρήση αυτής της επιλογής

πλακίδια και διευθύνσεις. Με αυτό τον τρόπο λειτουργεί ως διεπαφή ανάμεσα στην κωδικοποίηση που χρησιμοποιείται στην υπόλοιπη εφαρμογή και στην κωδικοποίηση που χρησιμοποιείται για την εύρεση κλειστών δωματίων.

- **Graph**
  Το αντικείμενο αυτό υλοποιεί λειτουργίες ενός συνεκτικού γράφου. Ο constructor της τάξης δέχεται ως όρισμα μια ακμή και στην ουσία δημιουργεί τους 2 πρώτους κόμβους. Οι βασικές λειτουργίες είναι η *attach()* η οποία δέχεται μια ακμή και προσπαθεί να τοποθετήσει τους κόμβους της στο γράφο με συνεκτικό τρόπο. Και η *count()* η οποία δέχεται ένα κόμβο και μετράει πόσες φορές ο κόμβος αυτός περιέχεται στον γράφο.
- **Position**
  Το αντικείμενο αυτό χρησιμοποιείται ως ένα κοινό σύστημα αναπαράστασης συντεταγμένων για την εφαρμογή. Ακόμα προσφέρει μεθόδους μετατροπής της μιας αναπαράστασης στην άλλη. Οι αναπαραστάσεις αυτές όπως αναφερθήκαμε και παραπάνω είναι η καρτεσιανή που περιέχει δύο μεταβλητές για την γραμμή και τη στήλη και η μονοδιάστατη (id).
- **Range**
  Το αντικείμενο χρησιμοποιείται για να δημιουργεί εύρη τιμών.
- **ShuffledRange**
  Το αντικείμενο αυτό χρησιμοποιείται για να δημιουργεί *"τυχαίως διατεταγμένα"* εύρη τιμών. Η τάξη αυτή κληρονομεί την Range και προσθέτει τη λειτουργία του τυχαίου ανακατέματος των τιμών.

5.1. Αλγόριθμος minimax

Ο σκοπός της παρούσας εργασίας θεωρούμε πως ήταν η υλοποίηση του αλγόριθμου minimax. Για τον αλγόριθμο αυτό αρχικά απαιτείται η δημιουργία ενός δέντρου. Το κάθε επίπεδο του δέντρου έχει για κόμβους όλες τις δυνατές κινήσεις των παικτών εναλλάξ. Έτσι ο κάθε κόμβος αναπαριστά μια κίνηση και τα παιδιά του κάθε κόμβου αναπαριστούν όλες τις δυνατές κινήσεις-απαντήσεις του αντίπαλου παίκτη στην κίνηση αυτή. Καθώς εκτελείται ο αλγόριθμος η αξιολόγηση της κάθε κίνησης από τους κατώτερους κόμβους ανεβαίνει προς τα πάνω. Η επιλογή γίνεται εναλλάξ από τον κάθε παίκτη. Έτσι ο αντίπαλος επιλέγει για τιμή του κάθε κόμβου την τιμή από τα παιδιά με την μικρότερη αξιολόγηση, ενώ ο παίκτης μας την τιμή με την μεγαλύτερη. Στο τέλος ο βασικός κόμβος περιέχει την μέγιστη δυνατή τιμή λαμβάνοντας υπόψιν την καλύτερη δυνατή κίνηση-απάντηση του αντιπάλου.



Σχήμα 3: *Διάγραμμα ροής της maxValue(), όπου γίνεται αναδρομική κλήση της minValue().*

Για τον κάθε κόμβο υλοποιήσαμε την τάξη Node ακολουθώντας τις οδηγίες της εκφώνησης. Πέρα από αυτές όμως χρειαστήκαμε και μια επιπλέον πληροφορία, που αφορά το μονοπάτι της επιλογής των κόμβων από τον αλγόριθμο. Δηλαδή, σε κάθε κόμβο αποθηκεύουμε επιπλέον ένα δείκτη προς τον κόμβο από τον οποίο έγινε η επιλογή της τιμής του. Με αυτό τον τρόπο ήμαστε σε θέση, μετά την εκτέλεση του αλγόριθμου, να γνωρίζουμε τόσο την τιμής της αξιολόγησης, όσο και την κίνηση από την οποία προήλθε.

Η υλοποίηση του minimax αλγόριθμου έγινε στην τάξη MinMaxPlayer. Η τάξη αυτή κληρονομεί την Player και μαζί και τις βασικές λειτουργίες όπως τους Constructors και το accessor - mutator idiom. Ακολουθώντας τον σχεδιασμό της HeuristicPlayer, υλοποιήσαμε την MinMaxPlayer, δίχως κανένα data member. Αυτά ανήκουν όλα στην Player. Έτσι όλη η επιπρόσθετη λειτουργικότητα της τάξης αφορά τον αλγόριθμο nimimax. Οι κύριες μέθοδοι που υλοποιήθηκαν για την δημιουργία του δέντρου είναι:

- ***int[] dryMove()***
  Η συνάρτηση αυτή προσομοιώνει την κίνηση κάποιου παίκτη χρησιμοποιώντας για ταμπλό αυτό από τα ορίσματά της. Έτσι μπορούμε να πραγματοποιήσουμε διαφορετικές κινήσεις σε διαφορετικούς κλώνους του ταμπλό χωρίς να αλλοιώσουμε το βασικό ταμπλό του παιχνιδιού.
- ***void createMySubtree()***
  Η συνάρτηση αυτή είναι η μία από τις δύο μεθόδους που υλοποιούν το δέντρο με τις δυνατές κινήσεις των παικτών. Ξεκινώντας από τη τρέχουσα θέση του παίκτη και για κάθε "διασχίσιμη διεύθυνση" δημιουργούμε και ένα αντίγραφο του ταμπλό και σε αυτό εκτελούμε την κίνηση του παίκτη εικονικά μέσω της συνάρτησης *dryMove()*. Έπειτα αποθηκεύουμε στο δέντρο την κίνηση αυτή ώς κόμβο και καλούμε την συνάρτηση *createOppSubtree()* με αυτόν σαν όρισμα.
- ***void createOppSubtree()***
  Η συνάρτηση εκτελεί την ίδια διαδικασία με την *createMySubtree()* αλλά για τον αντίπαλο αυτή τη φορά.
- ***double evaluation()***
  Η συνάρτηση αυτή είναι υπεύθυνη για την αξιολόγηση της κάθε κίνησης. Τόσο στην περίπτωση της του παίκτη, όσο και στην περίπτωση του αντιπάλου. Για να είναι συμβατή με τα ζητούμενα, αξιολογεί την κίνηση "κοιτώντας" το ταμπλό στην κατεύθυνση της κίνησης και μόνο. Αυτή η συμπεριφορά είναι αρκετά περιοριστική και οδηγεί σε πολύ φτωχά αποτελέσματα. Για παράδειγμα ενώ μια κίνηση προς τον αντίπαλο αξιολογείται αρνητικά καθώς ο παίκτης μπορεί να τον "δει", η κατεύθυνση μακριά από τον αντίπαλο δεν αξιολογείται θετικά. Ακόμα στην περίπτωση που η αξιολόγηση αφορά κίνηση του αντιπάλου, δεν έχουμε την δυνατότητα να αξιολογήσουμε την κατάσταση της πίστας συνολικά. Για να αντιμετωπίσουμε αυτόν τον τελευταίο περιορισμό, μετά την προσομοίωση της κίνησης του αντιπάλου, αξιολογούμε ξανά την τελευταία κίνηση του παίκτη. Μόνο που πλέον ο αντίπαλος είναι σε νέα θέση.

Τόσο η *createMySubtree()*, όσο και η *createOppSubtree()* εσωτερικά αξιολογούν την κάθε κίνηση. Αυτό είναι περιττό καθώς οι μόνοι κόμβοι που απαιτείται να αξιολογηθούν είναι τα φύλλα. Παρόλα αυτά αποφασίσαμε να εκτελέσουμε την *evaluate()* που αξιολογεί την κίνηση σε κάθε κόμβο ώστε να συμβαδίζουμε με τις οδηγίες της εκφώνησης.

Ο αλγόριθμος ολοκληρώνεται με την υλοποίηση των συναρτήσεων που ανάγουν τις αξιολογήσεις των κόμβων προς τα πάνω. Αυτές είναι οι *maxValue()* και *minValue()* και εκτελούνται αναδρομικά εναλλάξ από τον παίκτη και τον αντίπαλο. Στο σχήμα 3 φαίνεται το διάγραμμα ροής της *maxValue()*. Η *minValue()* είναι αντίστοιχη. Ο παίκτης προσπαθεί μέσω της *maxValue()* να επιλέξει τη μεγαλύτερη αξιολόγηση ενώ ο αντίπαλος, μέσω της *minValue()*, επιλέγει την μικρότερη.

6. Εκτελεσιμο

Όπως αναφέραμε και στην παράγραφο με τα παραδοτέα, σε αυτά υπάρχει και το παραγόμενο εκτελέσιμο για την γραμμή εντολών. Πρόκειται για ένα jar αρχείο το οποίο μπορεί κάποιος να εκτελέσει σε ένα τερματικό, μέσω της εντολής `java -jar labyrinth`. Στον κώδικά μας χρησιμοποιούμε asserions ώστε να ελέγξουμε την είσοδο και τις επιλογές του χρήστη. Επομένως αν κάποιος θέλει να πειραματιστεί με τις επιλογές καλό θα ήταν να τα ενεργοποιήσει στην VM της java με την παράμετρο `-ea`. Σε αυτή την περίπτωση θα μπορούσε να εκτελέσει `java -ea -jar labyrinth -i --norooms ...etc`

Το παραγόμενο jar παρέχει ένα αριθμό από επιλογές-ορίσματα τα οποία ελέγχουν την λειτουργία του παιχνιδιού. Τις επιλογές αυτές μπορούμε να δούμε και από την γραμμή εντολών απλώς εκτελώντας την εντολή `java -jar labyrinth -h`. Στο σχήμα 4 μπορούμε να δούμε ένα στιγμιότυπο με τις διαθέσιμες επιλογές-ορίσματα. Αναλυτικά, εκτός από την `-h` αυτές είναι:

- **-i** ή **--interactive**
  Αυτό το όρισμα ενεργοποιεί το *"interactive mode"*. Πρόκειται για λειτουργία κατά την οποία το παιχνίδι μετά από κάθε γύρο σταματά και περιμένει είσοδο από τον χρήστη για να προχωρήσει στον επόμενο. Αν αυτή η λειτουργία δεν είναι ενεργοποιημένη, τότε το παιχνίδι εκτελείται μονομιάς και

```
hoo2@shirka:~/Software/AUTH/Labyrinth/out$ java -jar labyrinth.jar -h
Labyrinth Game

Usage:
labyrinth [-b|--board <Num>] [-s|--supplies <Num>] [-r|--rounds <Num>] [--norooms] [-i|--interactive]
or
labyrinth -h|--help

Options

-b  | --board:
   Sets the size of board's edge.

-s  | --supplies:
   Sets the number of supplies on the board.

-r  | --rounds:
   Sets the maximum number of rounds of the game.

--norooms:
   Prevents the creation of closed rooms inside the board.

-i  | --interactive:
   Each round requires user input in order to continue.

-h  | --help:
   Print this and exits.

hoo2@shirka:~/Software/AUTH/Labyrinth/out$
```

Σχήμα 4: *Οι διαθέσιμες επιλογές και οι τρόποι με τους οποίους μπορούμε να εκτελέσουμε το παιχνίδι.*

στην έξοδο εκτυπώνονται όλοι οι γύροι αμέσως μετά την είσοδο της εντολής. *Αυτή η επιλογή είναι απενεργοποιημένη ως προεπιλογή.*

- **--norooms**
  Αυτό το όρισμα ενεργοποιεί την λειτουργία της εύρεσης και αποτροπής των κλειστών δωματίων. Αν δεν γίνει χρήση αυτή της επιλογής, τότε η λειτουργία αυτή είναι απενεργοποιημένη ως προεπιλογή. Η ενεργοποίησή της θεωρούμε πως είναι καλή επιλογή. Ο λόγος που δεν την προ-επιλέγουμε είναι γιατί δεν ζητείται ρητά από την εκφώνηση.
- **-b** ή **--board**
  Αυτό το όρισμα απαιτείται να ακολουθηθεί από ένα θετικό ακέραιο που περιγράφει το επιθυμητό μέγεθος της πλευράς του ταμπλό. Πχ:

  ```
  java -jar labyrinth -b 11
  ```

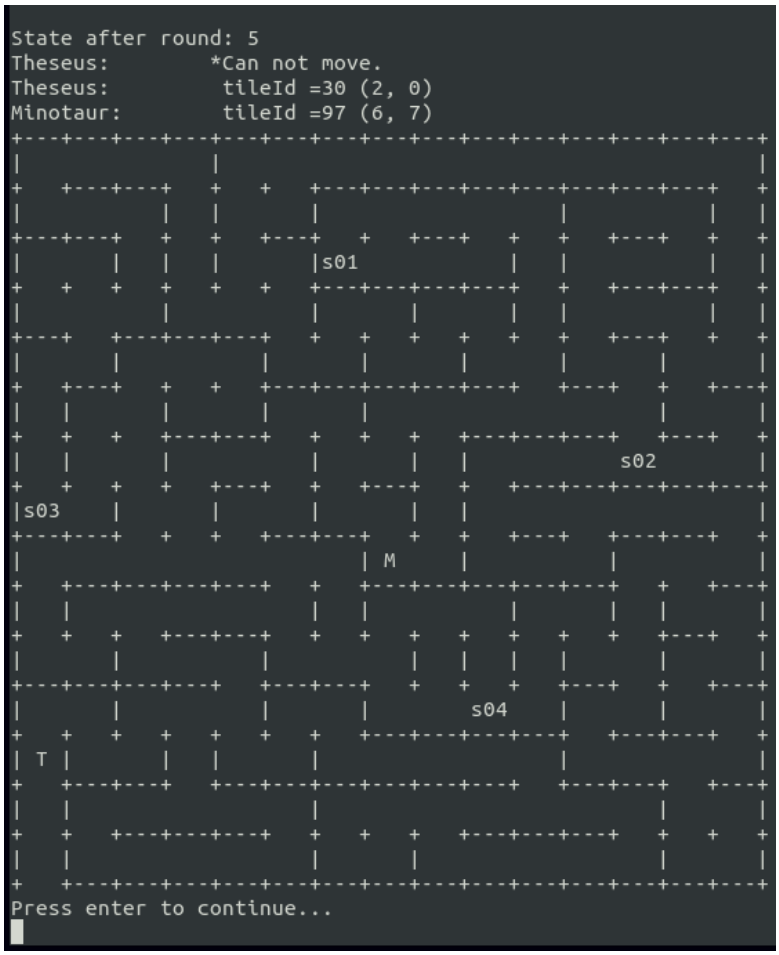
  Αν ο χρήστης δεν χρησιμοποιήσει την επιλογή τότε το προεπιλεγμένο μέγεθος είναι 15.
- **-s** ή **--supplies**
  Αυτό το όρισμα απαιτείται να ακολουθηθεί από ένα θετικό ακέραιο που περιγράφει τον επιθυμητό αριθμό των εφοδίων για το παιχνίδι. Πχ:

  ```
  java -jar labyrinth -b 11 -s 7
  ```

  Αν ο χρήστης δεν χρησιμοποιήσει την επιλογή τότε ο προεπιλεγμένος αριθμός είναι 4.
- **-r** ή **--rounds**
  Αυτό το όρισμα απαιτείται να ακολουθηθεί από ένα θετικό ακέραιο που περιγράφει τον επιθυμητό αριθμό γύρων μέχρι να ολοκληρωθεί το παιχνίδι. Πχ:

  ```
  java -jar labyrinth -r 200 -s 7
  ```

  Αν ο χρήστης δεν χρησιμοποιήσει την επιλογή τότε ο προεπιλεγμένος αριθμός είναι 100.



Σχήμα 5: *Στιγμιότυπο από την εκτέλεση του προγράμματος σε interactive mode και με αποφυγή κλειστών δωματίων. Η εντολή που χρησιμοποιήθηκε είναι* `java -jar labyrinth -i --norooms`

7. Παρατηρησεις

Κλείνοντας θα θέλαμε να πούμε πως η παρούσα εργασία ήταν μια προσπάθεια να υλοποιήσουμε τα ζητούμενα χρησιμοποιώντας όμως πρακτικές που θεωρούμε αναγκαίες για την συγγραφή κώδικα. Πρακτικές που στόχο έχουν να αποτρέψουν την πιθανότητα λανθασμένης χρήσης σε βάθος χρόνου. Για το λόγο αυτό προσπαθήσαμε να περιορίσουμε, όπου αυτό δεν ήταν αντίθετο με τα ζητούμενα, την δυνατότητα της λανθασμένης χρήσης εισάγοντας αρκετά επιπλέον εργαλεία για την εφαρμογή. Τέτοια για παράδειγμα είναι η τάξη Position ή η αποφυγή των setters κτλ. Ακόμα προσπαθήσαμε να οργανώσουμε τις δομές δεδομένων με τέτοιο τρόπο ώστε να περιορίσουμε τον απαιτούμενο κώδικά όσο το δυνατό περισσότερο. Αυτό έχει ως αποτέλεσμα ένα κάπως πιο μακροσκελές κώδικα, με πολλά αντικείμενα και συναρτήσεις, που ίσως είναι δυσανάλογος των μικρών απαιτήσεων της εργασίας. Ελπίζουμε να μην θεωρηθούν υπερβολικά.