ΑΡΙΣΤΟΤΕΛΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ

Τμημα ΗΜΜΥ. Τομεας Ηλεκτρονικης

Δομες δεδομενων

Λαβύρινθος: Ο Θησέας και ο Μινώταυρος 1

Συντάκτης :
Χρηστος Χουτουριδης
ΑΕΜ:8997
cchoutou@ece.auth.gr

Διδάσκον :
Σιαχαλου Σταυρουλα
ssiachal@auth.gr

26 Οκτωβρίου 2020

1. Εισαγωγη

Η παρούσα εργασία αφορά τη δημιουργία ενός παιχνιδιού λαβυρίνθου με θέμα *"Μια νύχτα στο μουσείο"*. Στο συγκεκριμένο παιχνίδι καλούμαστε να δημιουργήσουμε ένα λαβύρινθο μέσα στον οποίο κινούνται με τυχαίο τρόπο δύο παίχτες, ο Θησέας και ο Μινώταυρος. Στόχος του Μινώταυρου είναι να "πιάσει" τον Θησέα και στόχος του Θησέα είναι να βρει όλα τα εφόδια που είναι τυχαία κατανεμημένα στο ταμπλό πριν ξημερώσει και πριν τον πιάσει ο Μινώταυρος. Στο παιχνίδι αυτό υπάρχουν δύο βασικά προβλήματα τα οποία χρειάζεται να λύσουμε. Το πρόβλημα της δημιουργίας του λαβύρινθου και το πρόβλημα της λειτουργίας των παιχτών και του υπόλοιπου παιχνιδιού.

Κατά την άποψή μας η δημιουργία του ταμπλό είναι το κυριότερο από τα δύο προβλήματα. Το ταμπλό αποτελείται από πλακίδια και τοίχους. Τα πλακίδια είναι διατεταγμένα σε τετραγωνικό σχήμα και ανάμεσά τους τοποθετούνται οι τοίχοι. Το πρόβλημα έγκειται στην επιλογή και τοποθέτηση τοίχων με τέτοιο τρόπο ώστε να πληρούνται οι προδιαγραφές του παιχνιδιού όπως πχ κάθε πλακίδιο να μπορεί να έχει το πολύ δύο τοίχους ή τα εξωτερικά πλακίδια να έχουν τοίχο από την έξω μεριά. Μετά από μια πιο λεπτομερή ανάλυση του προβλήματος, διαπιστώσαμε πως οι δοθείσες προδιαγραφές δεν αποτρέπουν τη δημιουργία κλειστών δωματίων, κάτι που θεωρήσαμε άδικο, με αποτέλεσμα, όπως περιγράφουμε και αναλυτικά παρακάτω, **να προσθέσουμε έναν ακόμη περιορισμό. Την αποτροπή κλειστών δωματίων στο ταμπλό**.

Το πρόβλημα της λειτουργίας του υπόλοιπου παιχνιδιού έχει να κάνει με τη δημιουργία των παιχτών καθώς και τις κινήσεις τους. Οι προδιαγραφές αφορούν περιορισμούς στην κίνηση των παιχτών και τον τρόπο με τον οποίο λειτουργεί το παιχνίδι. Για παράδειγμα οι παίχτες δεν μπορούν να περάσουν μέσα από τοίχους, ή οι παίχτες κινούνται κατά ένα πλακίδιο τη φορά κλτ. Σε αντίθεση με τη λύση στο πρόβλημα του ταμπλό εδώ δεν απαιτήθηκαν ιδιαίτερες τεχνικές.

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. Αναφορά σε αυτό τον κατάλογο υπάρχει στο eclipse project.
- 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/index.html
- Ένας υποκατάλογος **report/** που περιέχει 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.

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. Εν αντιθέτως με το ιδίομα αυτό, **τα αντικείμενα που υλοποιούνται ως αφαιρέσεις μπορούν να προσφέρουν μεθόδους που εκτελούν κάποια λειτουργία, κρύβοντας τελείως τις εσωτερικές λεπτομέρειες τις υλοποίησης**. Αυτός είναι και ο δρόμος που διαλέξαμε για το σχεδιασμό του προγράμματος. Η κάθε τάξη 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 πουθενά μέσα στο πρόγραμμα.

3.2. Ενοποιημένο σύστημα συντεταγμένων

Στις προδιαγραφές της 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 ότι 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 απαιτούνται οι συντεταγμένες των εφοδίων 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 ποιο 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 πληροφορίες θέσης, αλλά αντίθετα η τάξη Tile θα είχε επιπρόσθετη πληροφορία για το αν υπάρχει 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.

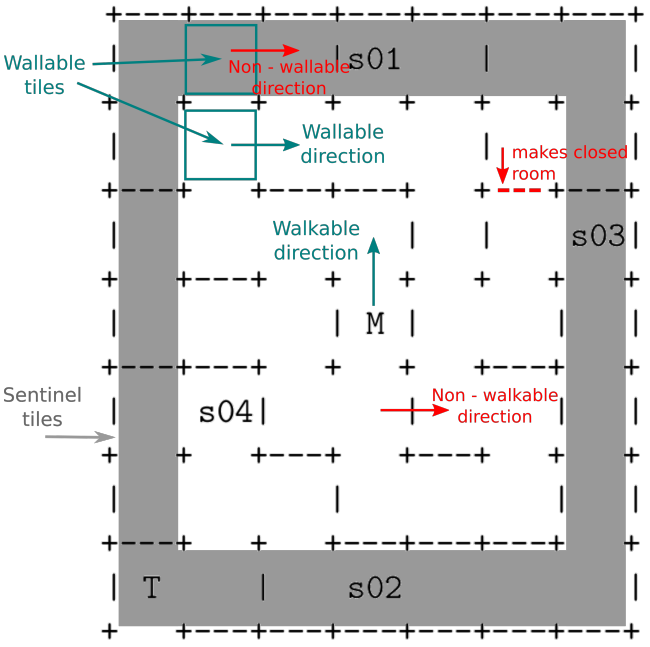
4. Concepts

Η δημιουργία της Board αποτέλεσε τον μεγαλύτερο όγκο του κώδικα της παρούσας εργασίας. Για να κάνουμε τον κώδικα καθαρότερο αλλά και ευκολότερο στην κατανόηση επινοήσαμε κάποια **concepts**. Η ιδέα των concepts προέρχεται από την C++ όπου τα concepts είναι ένα είδους compile time predicate και εφαρμόζεται στους τύπους δεδομένων. Εμείς για την εργασία υλοποιήσαμε κάποια concepts σε μορφή συναρτήσεων κατά την εκτέλεση του προγράμματος. Τα concepts αυτά αφορούν έννοιες σχετικές με την εφαρμογή και έχουν την μορφή predicate. Μας δίνεται έτσι η δυνατότητα να ελέγχουμε αν κάποια είσοδός ενός 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 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*

4.2. Διασχίσιμη διεύθυνση - *isWalkable()*

Πρόκειται για predicate που μας επιτρέπει να ελέγξουμε αν μια διεύθυνση σε κάποιο πλακίδιο είναι **"διασχίσιμη"**. 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.

4.3. Χτίσιμη διεύθυνση - *isWallableDir()*

Πρόκειται για predicate που μας επιτρέπει να ελέγξουμε αν μια διεύθυνση κάποιου πλακιδίου είναι **"χτίσιμη"**. 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 αριθμό πλακιδίων.
- Ο τοίχος να μην δημιουργεί κάποιο κλειστό δωμάτιο.
  Αυτή η τελευταία **η απαίτηση δεν υπάρχει στις προδιαγραφές** και την παίρνουμε υπόψιν 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. 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 πλακιδίο είναι **"χτίσιμο"**. Αν δηλαδή μπορούμε σε κάποια πλευρά του πλακιδίου να τοποθετήσουμε τοίχο. Για να ισχύει αυτό θα πρέπει:

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

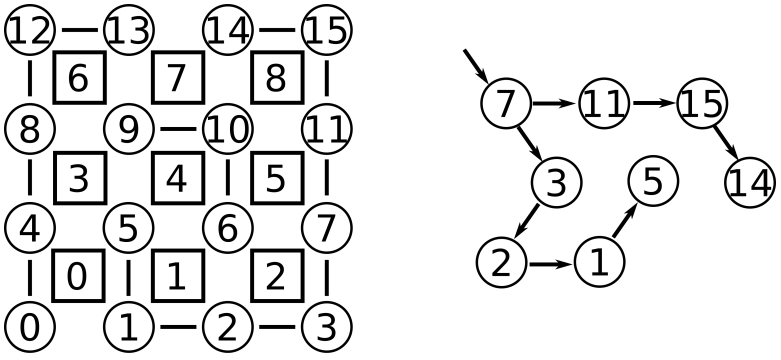
4.5. Δημιουργός κλειστού δωματίου - *isRoomCreator()*

Όπως αναφέραμε και παραπάνω οι προδιαγραφές της εργασίας δεν αποτρέπουν τη δημιουργία 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 διασταυρώσεις ανάμεσα στα πλακίδια ονομάζονται κόμβοι και η θέση τους αναπαρίστανται μονοδιάστατα σαν διεύθυνση σε row majer order.
- Ο κάθε κόμβος αποτελεί τον κόμβο-γωνία (vertex) ενός γράφου.
- Ο κάθε τοίχος αναπαρίσταται ως ακμή στον γράφο.

Για παράδειγμα ένας τοίχος αριστερά του πλακιδίου `0` αναπαρίσταται ως η ακμή `(0,4)`, Στο σχήμα 2 φαίνεται ένα παράδειγμα αναπαράστασης.



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 αλγόριθμο υλοποιήσαμε δύο αντικείμενα. Το **Edge** που δημιουργεί ζευγάρια κόμβων ώστε να μπορεί να αποθηκεύσει τον κάθε τοίχο(ακμή) και το **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 κάποιο τοίχο από τουλάχιστον δύο κατευθύνσεις. Άρα το ταμπλό περιέχει κάποιο κλειστό δωμάτιο. Αφού τον αλγόριθμο τον εκτελούμε για κάθε πιθανή χτίσιμη διεύθυνση, τότε ο τοίχος που προκαλεί το κλειστό δωμάτιο είναι αυτός που ελέγχουμε την εκάστοτε στιγμή.

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.

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 $O(N^2logN)$, όπου N, ο συνολικός αριθμός πλακιδίων του ταμπλό. Αυτό το κόστος αφορά τον έλεγχο του κάθε πλακιδίου. Για όλο το ταμπλό το κόστος είναι $O(N^4logN)$. 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 δημιουργία της εφαρμογής, εκτός από τα ζητηθέντα αντικείμενα υλοποιήσαμε και τα παρακάτω.

- **Const**
  Το αντικείμενο αυτό περιέχει σταθερές για όλη την εφαρμογή.
- **Session**
  Το αντικείμενο αυτό περιέχει όλες τις τιμές της εφαρμογής που αποτελούν 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 δημιουργία 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 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 αν κάποιος κόμβος της ακμής ανήκει ήδη στο γράφο τότε τοποθετεί και τον άλλο. Και η *count()* η οποία δέχεται ένα κόμβο και μετράει πόσες φορές ο κόμβος αυτός περιέχεται στον γράφο.
- **Position**
  Το αντικείμενο αυτό χρησιμοποιείται για να παρέχει ένα ενοποιημένο σύστημα συντεταγμένων για όλη την εφαρμογή.
- **Range**
  Το αντικείμενο χρησιμοποιείται για να δημιουργεί εύρη τιμών.
- **ShuffledRange**
  Το αντικείμενο αυτό χρησιμοποιείται για να δημιουργεί "τυχαίως ανακατεμένα" εύρη τιμών.

6. Εκτελεσιμο

7. Συμπερασματα και παρατηρησεις