ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

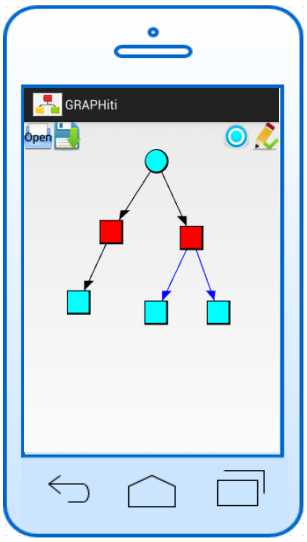
ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ ΚΑΙ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

Εφαρμογή Απεικόνισης Γραφημάτων σε συσκευές Android

Διπλωματική Εργασία

Λουκία Π. Κελίρη

ΑΥΓΟΥΣΤΟΣ 2014



Επιβλέπων: Αντώνιος Συμβώνης
Καθηγητής Ε.Μ.Π.

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ ΚΑΙ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

Εφαρμογή Απεικόνισης Γραφημάτων σε συσκευές Android

Διπλωματική Εργασία

Λουκία Π. Κελίρη

ΑΥΓΟΥΣΤΟΣ 2014

Επιβλέπων: Αντώνιος Συμβώνης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή:

……	……	……
Αντώνιος Συμβώνης	Κολέτσος Ιωάννης	Στεφανέας Πέτρος
Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.	Λέκτορας Ε.Μ.Π.

……
Λουκία Π. Κελίρη
Διπλωματούχος της Σχολής Εφαρμοσμένων Μαθηματικών και Φυσικών Επιστημών Ε.Μ.Π.

Περίληψη

Η απεικόνιση γραφημάτων μπορεί να διαδραματίσει πολύ σημαντικό ρόλο στην οπτικοποίηση διαφόρων πληροφοριών στα κινητά τηλέφωνα. Μέχρι στιγμής οι υπάρχουσες λύσεις είτε είναι σε αρχικά στάδια ή δίνουν έμφαση στο κομμάτι της οπτικοποίησης και δεν δίνουν ιδιαίτερη βαρύτητα στο γράφημα ως μαθηματική έννοια.

Σε αυτή την εργασία αναπτύχθηκε βιβλιοθήκη σε κώδικα Java, για απεικόνιση γραφημάτων σε συσκευές με λογισμικό Android, καθώς επίσης και η εφαρμογή GRAPHiti η οποία χρησιμοποιεί τη βιβλιοθήκη αυτή, και παρέχει ένα γραφικό περιβάλλον επικοινωνίας στο οποίο ο χρήστης μπορεί να δημιουργήσει ένα γράφημα, ή να φορτώσει κάποιο ήδη υπάρχον από αρχείο τύπου GraphML, να το επεξεργαστεί τροποποιώντας χαρακτηριστικά των κόμβων και των ακμών του, να διατάξει αυτόματα τους κόμβους του σε κύκλο, και τέλος, να το αποθηκεύσει στη συσκευή του.

Λέξεις κλειδιά

Γραφήματα, απεικόνιση γραφημάτων, βιβλιοθήκη, εφαρμογή Android, Android

Abstract

Graph drawing can play an important role in visualizing information on smartphones. So far, the existing solutions are either in early stages, or they emphasize on visualizing the graph and they are not paying much attention to the graph as a mathematical concept.

As part of this thesis, a java library has been developed for visualizing graphs in Android devices, as well as the GRAPHiti application which relies on this library and provides a Graphical User Interface (GUI), through which the user can create, edit, automatically arrange a graph in a circular layout and also, save and load GraphML files.

Keywords:

Graphs, drawing graphs, library, Android application, Android

Πίνακας Περιεχομένων

Ευχαριστίες 7

Κεφάλαιο 1: ΑΠΕΙΚΟΝΙΣΗ ΓΡΑΦΗΜΑΤΩΝ 9

1.1. Ορισμός Γραφήματος 9
1.2. Απεικόνιση γραφήματος και αισθητικά κριτήρια 9
1.3. Αλγοριθμικές προσεγγίσεις για την απεικόνιση γραφημάτων 12
1.4. Απεικόνιση Γραφημάτων σε κινητά τηλέφωνα 14

Κεφάλαιο 2: ΣΥΝΑΦΕΙΣ ΕΦΑΡΜΟΓΕΣ/ΒΙΒΛΙΟΘΗΚΕΣ 15

Κεφάλαιο 3: ΔΙΕΠΑΦΗ ΧΡΗΣΤΗ (UI) ΚΑΙ ΓΡΑΦΙΚΑ ΣΤΟ ANDROID 19

3.1. Διεπαφή χρήστη (User Interface) 19
3.1.1. Views και Layouts 19
3.2. Δισδιάστατα (2D) και Τρισδιάστατα (3D) Γραφικά στο Android 21
3.2.1. Σχεδιασμός με την κλάση Canvas 22

Κεφάλαιο 4: ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΒΙΒΛΙΟΘΗΚΗΣ 23

4.1. Βασική Δομή του Γραφήματος 23
4.2. Πρόσβαση στα στοιχεία του γραφήματος 24
4.3. «Αποθήκευση» των στοιχείων του γραφήματος 24
4.4. Δημιουργία γραφημάτων, κόμβων και ακμών 25
4.5. Διαγραφή κόμβων και ακμών 25
4.6. Περισσότερες λειτουργίες που παρέχει η κλάση Graph 26
4.7. Στοιχεία του γραφήματος 28
4.7.1. Κλάση Node 28
4.7.2. Κλάση Edge 29
4.8. Μηχανισμοί προσπέλασης 29
4.9. Χρήση HashMap για δέσμευση πληροφορίας στα στοιχεία του γραφήματος 30
4.10. Απεικόνιση και Επεξεργασία των γραφημάτων 31
4.11. Κλάσεις VisualGraph, VisualNodeRealizer, και VisualEdgeRealizer 31
4.11.1. Δημιουργία στοιχείων με προκαθορισμένες τιμές για τους realizers 32
4.11.2. Δημιουργία στοιχείων με δοθείσες τιμές για τους realizers 32
4.11.3. Node Realizers 33
4.11.4. Edge Realizers 34

4.12. Κλάση Label - Χρήση Ετικετών Κειμένου και Realizers 36
4.13. Αλληλεπίδραση με τον χρήστη και Listeners 36
4.13.1. Κλάση DefaultVisualEditMode 39
4.13.2. Κλάση DefaultVisualViewMode 41
4.14. Κλάση VisualGraphView 41
4.14.1. Προσθήκη εξατομικευμένου VisualModeListener σε ένα View 43
4.15. Αποθήκευση και ανάκτηση ενός γραφήματος 45
4.15.1. GraphML αρχείο 45
Κεφάλαιο 5: ΕΦΑΡΜΟΓΗ GRAPHiti 49
5.1. Δημιουργία κόμβων και ακμών 49
5.2. Επεξεργασία των στοιχείων του γραφήματος 51
5.3. Zooming και Panning 56
5.4. Αυτόματη διάταξη των κόμβων σε κύκλο 57
5.5. Αποθήκευση και Ανάκτηση ενός γραφήματος 59
Κεφάλαιο 6: ΣΥΝΟΨΗ 61
Βιβλιογραφία 63
Παράρτημα: Κώδικας της Εφαρμογής GRAPHiti 65

Ευχαριστίες

Στα πλαίσια της διπλωματικής μου εργασίας θα ήθελα να εκφράσω τις ειλικρινείς ευχαριστίες μου σε όλους όσους συνέβαλαν στην περάτωσή της.

Αρχικά θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, Δρ. Αντώνιο Συμβώνη για την εμπιστοσύνη που μου έδειξε με την ανάθεση αυτής της εργασίας, καθώς επίσης και για την καθοδήγηση και βοήθεια που μου προσέφερε καθ' όλη τη διάρκεια της εκπόνησής της.

Επίσης, ευχαριστώ θερμά τους γονείς μου για την υποστήριξη, τη συμπαράσταση και την υπομονή τους καθ' όλη τη διάρκεια των σπουδών μου, τον αδερφό μου για τις πολύτιμες συμβουλές που μου προσέφερε σε πολλές φάσεις των σπουδών μου, καθώς επίσης και τον Στέφανο για την υπομονή που έδειξε και την πολύτιμη βοήθειά του.

Τέλος, θα ήθελα να ευχαριστήσω όλους τους φίλους μου για τη συμπαράσταση τους στις δύσκολες στιγμές.

Κεφάλαιο 1: ΑΠΕΙΚΟΝΙΣΗ ΓΡΑΦΗΜΑΤΩΝ

1.1. Ορισμός Γραφήματος

Γράφημα είναι ένα διατεταγμένο ζεύγος δύο ξένων μεταξύ τους συνόλων V (≠Ø) και E, όπου το E είναι ένα υποσύνολο του συνόλου των μη διατεταγμένων ζευγών.
Τα στοιχεία του συνόλου V ονομάζονται κορυφές ή κόμβοι του γραφήματος και τα στοιχεία του συνόλου E, ακμές.
Όταν το γράφημα είναι κατευθυνόμενο τότε όλες οι ακμές του συνόλου ακμών του γραφήματος πρέπει να έχουν κατεύθυνση, δηλαδή ένα κόμβο-πηγή και ένα κόμβο-στόχο. Αυτό δηλώνεται μαθηματικά με ένα ζεύγος, όπου η πρώτη συνιστώσα υποδηλώνει τον κόμβο-πηγή και η δεύτερη τον κόμβο-στόχο.
Κανονικό γράφημα ονομάζεται το γράφημα στο οποίο κάθε ακμή ενώνει δύο κόμβους, ενώ σε κάθε άλλη περίπτωση το γράφημα ονομάζεται *υπεργράφημα*. Αν το ίδιο σύνολο κόμβων ενώνεται με πολλές ακμές, τότε οι ακμές αυτές ονομάζονται *πολλαπλές* ή *παράλληλες* και το γράφημα το οποίο τις περιλαμβάνει ονομάζεται *πολυγράφος.*

1.2. Απεικόνιση γραφήματος και αισθητικά κριτήρια

Πολλά προβλήματα παρουσίασης δεδομένων εμπλέκουν την απεικόνιση γραφημάτων. Κυρίως στις μέρες μας, με τα βελτιωμένα και ανεπτυγμένα γραφικά περιβάλλοντα επικοινωνίας, η απεικόνιση γραφημάτων έχει καταστεί ένας πολύ σημαντικός τομέας στην επιστήμη των υπολογιστών τόσο για τη διαχείριση δικτύων, όσο και για τη σχεδίαση βάσεων δεδομένων, το σχεδιασμό ιστοσελίδων και πολλά άλλα.
Ένας λόγος που κάποιος μπορεί να θέλει να σχεδιάσει ένα γράφημα, είναι η ανάγκη του ανθρώπου για οπτικοποίηση πληροφοριών όπως για παράδειγμα αντικείμενα και τις μεταξύ τους συσχετίσεις.
Επίσης, πολλά παραδείγματα από την καθημερινή ζωή αποτελούν εφαρμογές της απεικόνισης γραφημάτων, όπως για παράδειγμα οι χάρτες των γραμμών των τρένων και τα γενεαλογικά δέντρα.
Ως απεικόνιση γραφήματος ορίζουμε την εικονική αναπαράσταση των κόμβων και των ακμών ενός γραφήματος με σκοπό να δημιουργηθεί ένα αισθητικά ευχάριστο και ολοκληρωμένο σχεδιάγραμμα των δοθέντων δεδομένων. Αυτό βέβαια δε σημαίνει τη δημιουργία μιας στατικής εικόνας, αλλά γενικά περιγράφει μορφές αναπαράστασης με αλληλεπίδραση, επιτρέποντας δυναμικές αλλαγές στην εικόνα ενός γραφήματος.
Αξίζει να τονίσουμε ότι η απεικόνιση ενός γραφήματος δεν πρέπει να συγχέεται με

την αφηρημένη έννοιά του, αφού ένα γράφημα μπορεί να αναπαρασταθεί με πολλούς τρόπους.
Συνήθως οι κόμβοι ενός γραφήματος αναπαριστώνται σαν κύκλοι ή τετράγωνα και συμβολίζουν κάποιο αντικείμενο ή οντότητα, και οι ακμές (u, v) αναπαρίστανται σαν απλές ή πολυγωνικές γραμμές οι οποίες ενώνουν δύο κόμβους u και v και συμβολίζουν μία συσχέτιση μεταξύ των δύο αυτών κόμβων.
Οι πληροφορίες που αφορούν τους κόμβους και τις ακμές μπορούν να αναπαρασταθούν με τη χρήση ετικετών κειμένου σε διάφορες θέσεις όπως μέσα ή δίπλα από ένα συγκεκριμένο στοιχείο, σε διάφορα χρώματα, μορφοποιήσεις κτλ.
Ένα γράφημα μπορεί να αναπαρασταθεί στο δισδιάστατο ή τρισδιάστατο χώρο, και δύναται να έχει ιεραρχική δομή όπως συμβαίνει στην περίπτωση των συγκροτημάτων κόμβων όπου ένας κόμβος μπορεί να συρρικνωθεί και να επεκταθεί όταν χρειάζεται.
Οι τρόποι με τους οποίους θα χαρτογραφηθούν οι κόμβοι και οι ακμές ενός γραφήματος, στο επίπεδο ή στον τρισδιάστατο χώρο, ποικίλουν ανάλογα με τα κριτήρια και τους περιορισμούς κάθε περίπτωσης. Ως εκ τούτου χρειάζονται ειδικά διαμορφωμένοι αλγόριθμοι οι οποίοι κάθε φορά βελτιστοποιούν ένα ή περισσότερα από αυτά τα κριτήρια. Σε εφαρμογές όπου η απεικόνιση των γραφημάτων γίνεται με σκοπό αυτά να χρησιμοποιηθούν από ανθρώπους, τα κριτήρια αυτά ονομάζονται *αισθητικά κριτήρια*, ενώ υπάρχουν άλλες περιπτώσεις εφαρμογών όπου δίνεται περισσότερη έμφαση στα *τεχνικά κριτήρια* όπως για παράδειγμα στη σχεδίαση VLSI κυκλωμάτων.
Είναι σημαντικό να αναφερθεί ότι δίνοντας έμφαση σε διαφορετικά κριτήρια απεικόνισης, ένα γράφημα μπορεί να έχει πολλές διαφορετικές αναπαραστάσεις. Πιο κάτω θα αναφερθούν μερικά από τα πιο συχνά χρησιμοποιούμενα αισθητικά κριτήρια, με παραδείγματα από τα οποία διαφαίνεται η πρακτική σημασία τους.

Ελαχιστοποίηση διασταυρώσεων

Όταν πολλές ακμές διασταυρώνουν η μία την άλλη, το ανθρώπινο μάτι δυσκολεύεται να ξεχωρίσει ποιους κόμβους συνδέει μια συγκεκριμένη ακμή. Επομένως, είναι προτιμητέο όταν ένα γράφημα μπορεί να αναπαρασταθεί χωρίς διασταυρώσεις ακμών, να αποφεύγεται ο σχεδιασμός του με διασταυρώσεις. Γραφήματα τα οποία μπορούν να αναπαρασταθούν χωρίς διασταυρώσεις ακμών, ονομάζονται *επίπεδα γραφήματα* και ο σχεδιασμός τους δεν αποτελεί ιδιαίτερα δύσκολη διαδικασία.

Ελαχιστοποίηση σημείων καμπής

Η ελαχιστοποίηση σημείων καμπής είναι ένα σημαντικό αισθητικό κριτήριο για τις ορθογώνιες διατάξεις, γιατί το ανθρώπινο μάτι μπορεί πολύ πιο εύκολα να ακολουθήσει μία ακμή με καθόλου ή με ελάχιστα σημεία καμπής. Στη σχεδίαση

VLSI, τα σημεία καμπής σε καλώδια είναι πιθανές εστίες προβλημάτων, οπότε η ελαχιστοποίηση των σημείων καμπής είναι ένα σημαντικό τεχνικό κριτήριο.

Ελαχιστοποίηση χώρου

Η ελαχιστοποίηση του χώρου που καταλαμβάνει μία διάταξη είναι ένα σημαντικό κριτήριο στη σχεδίαση VLSI κυκλωμάτων, αλλά είναι επίσης και ένα γενικό αισθητικό κριτήριο επειδή η εικόνα ενός γραφήματος φαίνεται πολύ καλύτερη όταν οι κόμβοι και οι ακμές του γεμίζουν το χώρο ομοιόμορφα (με ομοιογενή πυκνότητα). Μια πιο προφανής περίπτωση όπου απαιτείται ελαχιστοποίηση χώρου, είναι για παράδειγμα η σχεδίαση χάρτη τσέπης για τις γραμμές των λεωφορείων.

Μεγιστοποίηση γωνίας

Το κριτήριο αυτό είναι ιδιαίτερα σημαντικό για απεικόνιση γραφημάτων σε συσκευές με μικρές οθόνες, όπως για παράδειγμα σε κινητά τηλέφωνα. Όταν ένα γράφημα απεικονίζεται σε οθόνη με χαμηλή ευκρίνεια, είναι σημαντικό οι ακμές να απέχουν όσο το δυνατόν περισσότερο.

Ελαχιστοποίηση μήκους

Στο σχεδιασμό VLSI κυκλωμάτων, οι ακμές απεικονίζουν καλώδια που μεταφέρουν πληροφορίες από ένα σημείο του τσιπ σε άλλο, και για να γίνονται οι μεταφορές όσο το δυνατόν συντομότερα, αυτά τα καλώδια πρέπει να είναι μικρού μήκους.

Συμμετρίες

Όταν ένα γράφημα περιέχει συμμετρική πληροφορία, τότε είναι σημαντικό να απεικονιστεί αυτή η συμμετρία στο γράφημα. Πολλά γραφήματα τεχνικής φύσης συχνά περιέχουν συμμετρίες. Δυστυχώς όμως, η απεικόνιση συμμετριών αποτελεί πολλές φορές ένα δύσκολο έργο.

Ομαδοποίηση

Κατά τη σχεδίαση γραφημάτων κοινωνικών δικτύων, και γενικότερα κατά τη σχεδίαση μεγάλων δικτύων και γραφημάτων, είναι σημαντική η ομαδοποίηση των κόμβων για την καλύτερη παρουσίαση της δομής του γραφήματος.

Πολυεπίπεδες απεικονίσεις

Στα οργανογράμματα, στα διαγράμματα οντοτήτων-συσχετίσεων και στα διαγράμματα ροής, συνήθως απαιτείται πολυεπίπεδη διάταξη, όπου οι θέσεις των κόμβων περιορίζονται σε διακριτά επίπεδα.

1.3. Αλγοριθμικές προσεγγίσεις για την απεικόνιση γραφημάτων

Μετά τον καθορισμό του πώς θα αναπαρασταθεί το γράφημα και του κατάλληλου συνδυασμού των αισθητικών κριτηρίων που θα χρησιμοποιηθούν, συνήθως προκύπτουν δύο άλλα προβλήματα.
Το πρώτο πρόβλημα που προκύπτει είναι ότι πολλά από τα κριτήρια δεν μπορούν να βελτιστοποιηθούν αποτελεσματικά και έτσι γίνεται συμβιβασμός με προσεγγιστικούς ή ευριστικούς (heuristics) αλγορίθμους.
Το δεύτερο πρόβλημα που προκύπτει είναι ότι μερικές φορές η βελτιστοποίηση πολλών κριτηρίων ταυτόχρονα είναι ακατόρθωτη γιατί τυχαίνει τα κριτήρια να έρχονται σε αντίθεση μεταξύ τους.
Για την αντιμετώπιση αυτών των προβλημάτων βελτιστοποίησης, υπάρχουν μερικές τεχνικές οι οποίες περιγράφονται σε συντομία παρακάτω.

Επιπεδοποίηση
Οι επίπεδες διατάξεις είναι συνήθως πιο ευανάγνωστες από τις μη επίπεδες. Επίσης, στο σχεδιασμό κυκλωμάτων οι τεχνικές για επιπεδοποίηση είναι πολύ σημαντικές στην ελαχιστοποίηση των στρωμάτων.
Δυστυχώς όμως στην πράξη πολλά γραφήματα είναι μη επίπεδα. Μπορούν να μετατραπούν σε επίπεδα είτε με αφαίρεση όσο λιγότερων ακμών γίνεται, το οποίο είναι NP-complete πρόβλημα, ή με επιλεκτική αφαίρεση ακμών των οποίων η μετέπειτα εισαγωγή θα δημιουργήσει το μικρότερο αριθμό διασταυρώσεων. Το πρόβλημα της ελαχιστοποίησης των διασταυρώσεων είναι γενικά ένα NP-hard πρόβλημα, αλλά υπάρχουν μερικοί ευριστικοί αλγόριθμοι οι οποίοι καταλήγουν σε αποδεκτά αποτελέσματα.

Force-directed μέθοδοι
Τέτοιου είδους αλγόριθμοι χρησιμοποιούνται για το σχεδιασμό αυθαίρετων (αραιών) δικτύων, όπως για παράδειγμα διαγράμματα ροής, και μπορούν επίσης να εφαρμοστούν σε διατάξεις συστάδων (clustered layouts).

Sugiyama-like μέθοδοι
Οι αλγόριθμοι τύπου Sugiyama είναι οι πιο συχνά χρησιμοποιούμενοι αλγόριθμοι για σχεδιασμό πολυεπίπεδων διατάξεων. Οι αλγόριθμοι αυτοί, προσπαθούν να ελαχιστοποιήσουν τον αριθμό των διασταυρώσεων ή το χώρο που καταλαμβάνει μια διάταξη.

Μέθοδοι ροής

Η ελαχιστοποίηση των σημείων καμπής μπορεί να επιλυθεί αποτελεσματικά με την αναγωγή του προβλήματος σε πρόβλημα ροής δικτύου. Οι ίδιες τεχνικές μπορούν να χρησιμοποιηθούν για τη μεγιστοποίηση των γωνιών μεταξύ ακμών.

Διαδραστικοί σχεδιασμοί

Εδώ συγκαταλέγονται τεχνικές οι οποίες είναι σημαντικές στο σχεδιασμό γραφημάτων οντοτήτων-συσχετίσεων, VLSI κυκλωμάτων, WWW γράφων, και άλλων μη στατικών γραφημάτων που αλλάζουν με την πάροδο του χρόνου.

Επισήμανση

Μία σημαντική πτυχή της απεικόνισης γραφημάτων είναι η προσθήκη ετικετών κειμένου στους κόμβους και τις ακμές, όπως παρατηρείται για παράδειγμα στο σχεδιασμό χαρτών, διαγραμμάτων καταστάσεων (state diagrams), ή μηχανικών διαγραμμάτων.

Αξίζει να σημειωθεί ότι τα αισθητικά κριτήρια πολλές φορές έρχονται σε αντίθεση με πιο διαισθητικά κριτήρια που αφορούν στο τι συμβολίζει το γράφημα. Επίσης, η σημασιολογία και η δομή ενός γραφήματος μπορεί να δώσει πολύ διαφορετικές επιλογές για τη διάταξή του. Επιπρόσθετα, η πλήρης παράβλεψη μερικών κριτηρίων, όπως για παράδειγμα το μήκος των ακμών που είναι ένα καθαρά τεχνικό κριτήριο, μπορεί να οδηγήσει σε κακοσχεδιασμένες και δυσανάγνωστες διατάξεις. Εναπόκειται λοιπόν στη φύση κάθε προβλήματος το αν και ποιοι αλγόριθμοι διάταξης θα χρησιμοποιηθούν στην αναπαράσταση ενός γραφήματος.

1.4. Απεικόνιση Γραφημάτων σε κινητά τηλέφωνα

Στη σύγχρονη εποχή ο αριθμός των ανθρώπων που χρησιμοποιούν smartphones ολοένα και αυξάνεται. Μία συσκευή με τόσο ευρεία χρήση μπορεί να αξιοποιηθεί για άμεση πρόσβαση σε ποικίλα είδη δεδομένων που αφορούν ποικιλία θεμάτων, από τα οποία ένα μεγάλο μέρος είναι σχεσιακές πληροφορίες. Για παράδειγμα τα κινητά τηλέφωνα χρησιμοποιούνται για πρόσβαση σε κοινωνικά δίκτυα όπως το Facebook, για πρόσβαση σε οντολογίες όπως το δίκτυο εννοιών της Wikipedia, καθώς επίσης και για πρόσβαση σε τεχνικές πληροφορίες που σχετίζονται με την εργασία του κατόχου της συσκευής όπως για παράδειγμα τις συνδέσεις ενός δικτύου υπολογιστών ή τις διαδρομές παράδοσης ενός συστήματος διανομής προϊόντων.

Η απεικόνιση γραφημάτων μπορεί να διαδραματίσει πολύ σημαντικό ρόλο στην οπτικοποίηση διαφόρων πληροφοριών στα κινητά τηλέφωνα, θα πρέπει όμως πρώτα να τροποποιηθούν οι ήδη υπάρχουσες μεθοδολογίες και τα εργαλεία που χρησιμοποιούνται γενικά στο πεδίο της απεικόνισης γραφημάτων, έτσι ώστε να ικανοποιούν τις ανάγκες μίας τέτοιας συσκευής.

Το κυριότερο πρόβλημα που καλούνται να αντιμετωπίσουν οι εφαρμογές απεικόνισης γραφημάτων στα κινητά τηλέφωνα είναι το μικρό μέγεθος της οθόνης. Αυτός ο τόσο σημαντικός περιορισμός ήταν ο λόγος που οδήγησε σε ποικίλες τεχνολογικές βελτιώσεις, όπως για παράδειγμα multi-touch οθόνες και αισθητήρες όπως επιταχυνσιόμετρα (accelerometers) και πυξίδες (compasses), με τα οποία βελτιώθηκε σημαντικά η αλληλεπίδραση με το χρήστη.

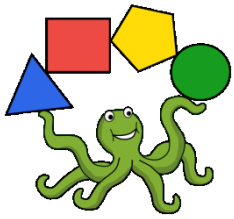
Η απεικόνιση γραφημάτων σε κινητά τηλέφωνα έχει αρκετό περιθώριο εξέλιξης, γιατί όπως αναφέρθηκε λόγω των μικρών διαστάσεων της οθόνης παρουσιάζονται νέα προβλήματα σχεδίασης/απεικόνισης και η ανάπτυξη μιας πιο ολοκληρωμένης εφαρμογής βρίσκεται ακόμα σε σχετικά αρχικά στάδια.

Κεφάλαιο 2: ΣΥΝΑΦΕΙΣ ΕΦΑΡΜΟΓΕΣ/ΒΙΒΛΙΟΘΗΚΕΣ

Στο κεφάλαιο αυτό παρουσιάζονται κάποιες ήδη υπάρχουσες εφαρμογές ή βιβλιοθήκες του Android, οι οποίες έχουν ως στόχο την απεικόνιση γραφημάτων/διαγραμμάτων σε κινητά τηλέφωνα.

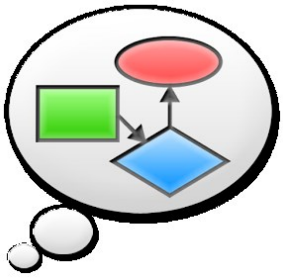
- DroidDia

Εφαρμογή στο Android η οποία επιτρέπει τη δημιουργία διαγραμμάτων ροής, διαγραμμάτων οργάνωσης, Venn διαγραμμάτων και πολλών άλλων διαγραμμάτων, χωρίς ιδιαίτερους περιορισμούς. Γενικά παρέχει παρόμοιες λειτουργίες με το Microsoft Visio, αλλά απευθύνεται σε συσκευές Android. Υπάρχει η έκδοση DroidDia prime η οποία είναι δωρεάν, αλλά και η DroidDia PRO unlocker με επιπρόσθετες λειτουργίες, η οποία είναι επί πληρωμή.



- Oqto

Εφαρμογή για Android tablet και κινητά τηλέφωνα, διαθέσιμη σε δύο εκδόσεις, τις Oqto Diagram και Oqto Diagram Pro η οποία είναι επί πληρωμή. Παρέχει σχεδιασμό διαγραμμάτων διαφόρων ειδών, και αποθήκευσή τους σε διάφορους τύπους αρχείων όπως PNG, JPEG, SVG και OQTO.



- Smart Diagram Lite

Εφαρμογή για σχεδιασμό διαγραμμάτων όπως για παράδειγμα διαγράμματα ροής, διαγράμματα δενδρικής δομής και πολλά άλλα, με την δυνατότητα αποθήκευσής τους σε αρχεία JPG, PNG και XML.
Έχει το μειονέκτημα ότι υποστηρίζεται από πολύ λίγες συσκευές.

- Lucidchart

Ένα **online** πρόγραμμα σχεδίασης γραφημάτων, διαγραμμάτων ροής, UML διαγραμμάτων, δικτύων, κτλ, το οποίο υποστηρίζει και άνοιγμα αρχείων του Microsoft Visio. Παρέχει επίσης τη δυνατότητα για ταυτόχρονο σχεδιασμό και τροποποίηση ενός διαγράμματος από ομάδα χρηστών. Παρόλο που έχει βελτιστοποιηθεί έτσι ώστε να μπορεί να χρησιμοποιείται online και από χρήστες Android, προς το παρόν λειτουργεί μόνο σε tablet γιατί ακόμα δεν είναι λειτουργικό σε μικρότερες οθόνες όπως αυτές των κινητών τηλεφώνων.

- yEd Graph Editor

Ο yEd Graph Editor είναι μια εφαρμογή για υπολογιστές η οποία μπορεί να χρησιμοποιηθεί για τη δημιουργία γραφημάτων υψηλής ποιότητας είτε με το χέρι είτε εισάγοντας κάποιο αρχείο. Επίσης παρέχει πολλά είδη αλγορίθμων για αυτόματη διάταξη των στοιχείων ενός γραφήματος. Η αντίστοιχη εφαρμογή του συγκεκριμένου editor για Android βρίσκεται σε εξέλιξη. Υπόσχεται να παρέχει υψηλής ποιότητας αλληλεπίδραση του χρήστη με το γράφημα, τόσο κατά την προβολή όσο και κατά την επεξεργασία, καθώς και αυτόματη διάταξη σύνθετων γραφημάτων, διαγραμμάτων και δικτύων. Μέχρι στιγμής υπάρχει διαθέσιμη δωρεάν η εφαρμογή **yWorks OrgChart Editor**, η οποία αποτελεί μία δοκιμαστική (demo) εφαρμογή που επιδεικνύει σε αρχικό στάδιο τις δυνατότητες της εφαρμογής.

- MindFusion.Diagramming για Android/ DroidDiagram

Είναι μία **βιβλιοθήκη** βασισμένη σε java η οποία μπορεί να χρησιμοποιηθεί για την απεικόνιση διαφόρων ειδών διαγραμμάτων. Περιέχει κλάσεις που αφορούν τη δομή και την εμφάνιση ενός διαγράμματος, καθώς και κλάσεις οι οποίες είναι υπεύθυνες για την απεικόνιση των διαγραμμάτων στην οθόνη. Οι κλάσεις αυτές μπορούν να χρησιμοποιηθούν σε οποιαδήποτε εφαρμογή του Android framework. Παρέχει επίσης ένα διαισθητικό μοντέλο αλληλεπίδρασης με τον χρήστη για τη δημιουργία ή την επεξεργασία διαγραμμάτων. Επιπρόσθετα, παρέχει προγραμματιστικά πολλές χρήσιμες λειτουργίες, όπως για παράδειγμα την

εκτύπωση ενός διαγράμματος ή την εξαγωγή του σε PDF, καθώς επίσης και αλγορίθμους αυτόματης διάταξης.

Το κυριότερο μειονέκτημα των περισσότερων διαθέσιμων εφαρμογών που αναφέρθηκαν πιο πάνω, οι οποίες υποστηρίζουν σχεδιασμό και αποθήκευση διαγραμμάτων σε συσκευές Android, είναι ότι δεν δίνεται σημασία στην έννοια του γραφήματος ως μαθηματική έννοια. Κύριος στόχος τους είναι να απεικονιστούν οι πληροφορίες στη μορφή ενός διαγράμματος, αλλά δεν διατηρείται η πληροφορία για τη δομή του γραφήματος.

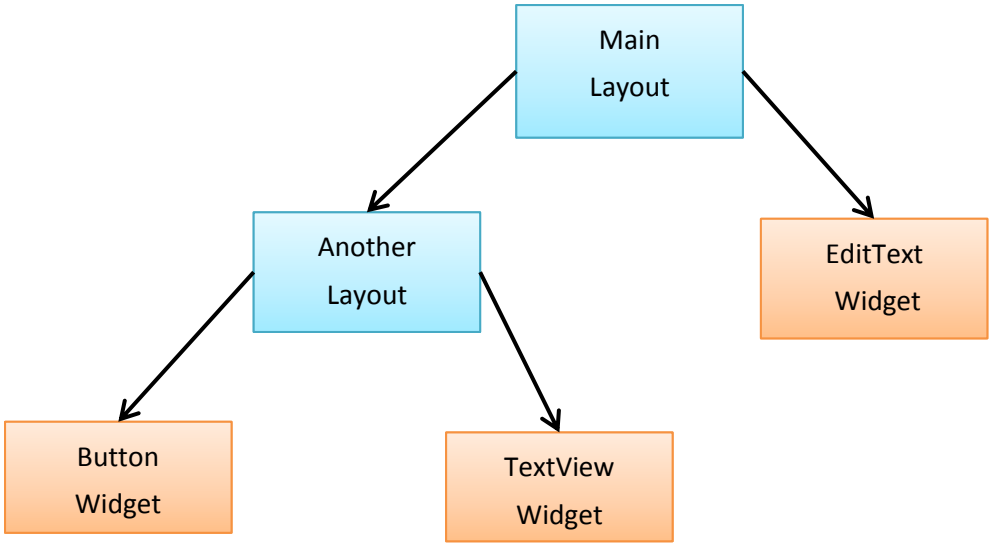
Κεφάλαιο 3: ΔΙΕΠΑΦΗ ΧΡΗΣΤΗ (UI) ΚΑΙ ΓΡΑΦΙΚΑ ΣΤΟ ANDROID

3.1. Διεπαφή χρήστη (User Interface)

Υπάρχουν δύο τρόποι δημιουργίας διεπαφής χρήστη, είτε προγραμματιστικά ή δηλωτικά.
Η προγραμματιστική δημιουργία περιλαμβάνει κώδικα σε Java. Για παράδειγμα, για τη δημιουργία ενός κουμπιού προγραμματιστικά, θα πρέπει αυτό να δηλωθεί ως μεταβλητή, να δημιουργηθεί, να προστεθεί σε ένα "container" και να καθοριστούν τα διάφορα χαρακτηριστικά του, όπως για παράδειγμα το χρώμα, το κείμενο που θα περιέχει, το μέγεθος του κειμένου και άλλα. Επίσης θα πρέπει να γραφεί ο κώδικας για το τι ενέργεια θα πραγματοποιείται όταν αυτό επιλεχθεί.
Η δημιουργία διεπαφής δηλωτικά, περιλαμβάνει τη χρήση XML αρχείου στο οποίο δηλώνεται το πώς θα μοιάζει η διεπαφή, και χρησιμοποιείται κυρίως για στοιχεία της διεπαφής τα οποία είναι στατικά, όπως για παράδειγμα το "layout" της οθόνης και τα διάφορα widgets.
Γενικά, για τη δημιουργία μίας διεπαφής χρήστη καλό είναι να χρησιμοποιούνται και οι δύο τρόποι. Με το XML αρχείο δηλώνονται όπως αναφέρθηκε και πιο πάνω τα στατικά στοιχεία της διεπαφής, ενώ προγραμματιστικά δηλώνονται τα κουμπιά καθώς και οι διάφορες λειτουργίες που θα πρέπει να εκτελούνται όταν ο χρήστης αλληλεπιδρά με την εφαρμογή.

3.1.1. Views και Layouts

Το Android οργανώνει τα στοιχεία της διεπαφής χρήστη σε "layouts" και "views". Ένα view είναι οτιδήποτε μπορούμε να δούμε, όπως για παράδειγμα τα κουμπιά και οι ετικέτες κειμένου (labels). Τα layouts οργανώνουν τα views, π.χ. ομαδοποιούν ένα κουμπί με μία ετικέτα κειμένου. Θα μπορούσαμε να αντιστοιχίσουμε τα views με τα containers στην Java, και τα layouts με τα components.
Ένα layout μπορεί να έχει «παιδιά» τα οποία μπορεί να είναι κι αυτά layouts, επιτρέποντας έτσι μία σύνθετη δομή στη διεπαφή χρήστη. Επίσης ένα layout είναι υπεύθυνο για την κατανομή χώρου σε κάθε παιδί του. Διαφορετικά είδη layouts έχουν διαφορετικές προσεγγίσεις σχετικά με την κατανομή του χώρου στα παιδιά τους.



Εικόνα 1 Παράδειγμα σχέσης view και layout.

Τα layouts τα οποία χρησιμοποιούνται συχνότερα είναι:

- ***LinearLayout***
  Αποτελεί το πιο απλό και κοινό είδος. Θέτει τα παιδιά του το ένα δίπλα στο άλλο, είτε οριζόντια είτε κάθετα. Η σειρά των παιδιών παίζει ρόλο αφού ο χώρος που απομένει στο κάθε παιδί εξαρτάται από τη σειρά με την οποία θα προστεθούν τα παιδιά.
- ***FrameLayout***
  Τοποθετεί τα παιδιά του το ένα στην κορυφή του άλλου, έτσι ώστε το τελευταίο να καλύπτει το προηγούμενο, όπως μία στοίβα από κάρτες.
- ***RelativeLayout***
  Καθορίζει τα παιδιά του το ένα σε σχέση με το άλλο. Είναι σημαντικό ότι με τη χρήση αυτού του είδους layout δε χρειάζεται η χρήση εμφωλευμένων layouts για την επίτευξη μίας συγκεκριμένης διάταξης. Επίσης ελαχιστοποιεί το συνολικό αριθμό των widgets της εφαρμογής που πρέπει να σχεδιαστούν, με αποτέλεσμα την καλύτερη απόδοση της εφαρμογής. Για τη συσχέτιση όμως των παιδιών μεταξύ τους θα πρέπει να διατηρείται ένα πεδίο ID για κάθε παιδί.
- ***AbsoluteLayout***
  Τοποθετεί τα παιδιά του σε απόλυτες συντεταγμένες μιας οθόνης. Δεν είναι πολύ εύχρηστο γιατί δεν μπορεί να προσαρμοστεί σε διαφορετικές οθόνες.

3.2. Δισδιάστατα (2D) και Τρισδιάστατα (3D) Γραφικά στο Android

Το Android παρέχει μεγάλη ποικιλία APIs (*Application Programming Interfaces*) για σχεδιασμό δισδιάστατων και τρισδιάστατων γραφικών.

Κατά τη δημιουργία μίας εφαρμογής είναι σημαντικό να εξετασθούν και να καθοριστούν με ακρίβεια οι γραφικές της απαιτήσεις. Διαφορετικές απαιτήσεις επιτυγχάνονται με διαφορετικές τεχνικές. Για παράδειγμα, τα γραφικά και τα animations μιας στατικής εφαρμογής υλοποιούνται πολύ διαφορετικά απ' ότι τα γραφικά και animations μίας διαδραστικής εφαρμογής.
Μερικές από τις επιλογές τις οποίες παρέχει το Android για το σχεδιασμό γραφικών είναι:

- **Canvas και Drawables**
  Το Android παρέχει ένα σύνολο από "view widgets" τα οποία καλύπτουν διάφορες λειτουργίες για ένα ευρύ φάσμα διεπαφών χρήστη. Επιπλέον, με τη χρήση διαφόρων μεθόδων σχεδίασης της κλάσης Canvas ή με τη δημιουργία Drawable αντικειμένων, υπάρχει η δυνατότητα εξατομικευμένης δισδιάστατης σχεδίασης.
- **Hardware Acceleration**
  Από την έκδοση Android 3.0 και έπειτα, η πλειοψηφία των σχεδιασμών που επιτυγχάνονται με το Canvas API μπορούν να επιταχυνθούν με hardware acceleration για ακόμη καλύτερες επιδόσεις.
- **Open GL**
  Το Android υποστηρίζει OpenGL ES 1.0 και 2.0, με χρήση Android framework APIs, καθώς επίσης και εγγενώς με το Native Development Kit (NDK). Η χρήση των framework APIs είναι επιθυμητή όταν χρειάζεται η προσθήκη γραφικών βελτιώσεων στην εφαρμογή, οι οποίες δεν υποστηρίζονται με τα Canvas APIs.

Στη βιβλιοθήκη που αναπτύχθηκε σ' αυτή τη διπλωματική εργασία, για το σχεδιασμό των γραφικών χρησιμοποιήθηκε η κλάση Canvas, της οποίας μερικά χαρακτηριστικά περιγράφονται στην επόμενη ενότητα.

3.2.1. Σχεδιασμός με την κλάση Canvas

Υπάρχουν δύο τρόποι με τους οποίους μπορούν να σχεδιαστούν δισδιάστατα γραφικά στο Android:

1) Σχεδιασμός των γραφικών σε ένα αντικείμενο της κλάσης View. Σ' αυτή την περίπτωση η σχεδίαση των γραφικών καθορίζεται από την κανονική ιεραρχία της κλάσης View.
2) Σχεδιασμός των γραφικών κατ' ευθείαν στο Canvas. Σ' αυτή την περίπτωση είτε καλείται η μέθοδος onDraw() της κατάλληλης κλάσης, ή μία από τις draw μεθόδους που παρέχει η κλάση Canvas, όπως για παράδειγμα η μέθοδος drawPicture().

Ο πρώτος τρόπος χρησιμοποιείται σε περιπτώσεις εφαρμογών οι οποίες έχουν απλά και στατικά γραφικά, ενώ ο δεύτερος τρόπος αφορά εφαρμογές όπως είναι τα βιντεοπαιχνίδια, όπου τα γραφικά τους αλλάζουν δυναμικά και απαιτείται επανασχεδίαση σε τακτά χρονικά διαστήματα.
Η κλάση Canvas του Android αναπαριστά μία επιφάνεια στην οποία σχεδιάζονται τα γραφικά, και χρησιμοποιείται σε εφαρμογές που απαιτούν εξειδικευμένο σχεδιασμό ή/και έλεγχο των animations. Μέσω του Canvas στην ουσία το σχέδιο εκτελείται επί ενός Bitmap, το οποίο τοποθετείται στο παράθυρο.

Μέσα στην κλάση Canvas περιέχονται μέθοδοι οι οποίες παρέχουν:

- Σχεδιασμό βασικών σχημάτων όπως τετραγώνων, ελλείψεων, κύκλων κτλ
- Σχεδιασμό Bitmaps
- Σχεδιασμό κειμένου
- Τροποποίηση του canvas (translate, scale)

Κλάση Paint

Για το σχεδιασμό σε ένα αντικείμενο Canvas είναι απαραίτητη η χρήση ενός αντικειμένου της κλάσης Paint. Η κλάση αυτή καθορίζει ιδιότητες για το σχεδιασμό των γραφικών οι οποίες αφορούν τα χρώματα, το πάχος των γραμμών κτλ.

Πιο κάτω φαίνεται ένα παράδειγμα σχεδιασμού ενός τετραγώνου σε ένα canvas.

```
// canvas is of type 'Canvas'
Paint paint = new Paint();
paint.setStyle(Style.FILL);
paint.setColor(Color.BLUE);
canvas.drawRect(0, 0, 10, 10,paint);
```

Κεφάλαιο 4: ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΒΙΒΛΙΟΘΗΚΗΣ

Σ' αυτό το κεφάλαιο γίνεται μία παρουσίαση της βιβλιοθήκης που αναπτύχθηκε, των πακέτων που περιλαμβάνει καθώς και της λειτουργικότητας αυτών.

Η λειτουργικότητα που παρέχει η βιβλιοθήκη χωρίζεται σε δύο μέρη:

- Basic, το βασικό κομμάτι για το κύριο μέρος της βιβλιοθήκης το οποίο περιέχει ουσιώδεις κλάσεις και τύπους δεδομένων για την ανάλυση ενός γραφήματος
- Visual, περιλαμβάνει οτιδήποτε σχετίζεται με την αλληλεπίδραση με τον χρήστη και βασίζεται στο Basic κομμάτι

4.1. Βασική Δομή του Γραφήματος

Η δημιουργία αυτής της βιβλιοθήκης στηρίζεται στη μαθηματική έννοια του γραφήματος, σύμφωνα με την οποία ένα γράφημα αποτελείται από δύο σύνολα, το σύνολο των κόμβων και το σύνολο των ακμών.
Αυτή η δομή επιτυγχάνεται με τις κλάσεις Graph, Node και Edge στο πακέτο basic, οι οποίες και αποτελούν το θεμέλιο για τη δημιουργία ενός γραφήματος σαν οντότητα.
Μερικές πτυχές της δομής του γραφήματος στη βιβλιοθήκη:

- Ένα γράφημα μπορεί να έχει κατευθυνόμενες ή μη κατευθυνόμενες ακμές
- Δεν υποστηρίζεται η δημιουργία self-loop ακμών
- Δεν υποστηρίζεται η δημιουργία παράλληλων ακμών, δηλαδή μόνο μία ακμή μπορεί να δημιουργηθεί μεταξύ δύο συγκεκριμένων κόμβων του γραφήματος

Αξίζει να σημειωθεί ότι η βιβλιοθήκη στηρίζεται στην ιδέα ότι υπεύθυνη για τις αλλαγές στη δομή (κυρίως σε ότι αφορά δημιουργία ή διαγραφή κόμβων και ακμών) είναι μόνο η οντότητα γράφημα. Δηλαδή ένας κόμβος ή μία ακμή δεν μπορεί να δημιουργηθεί «εκτός» γραφήματος, καθώς επίσης όλοι οι κόμβοι και οι ακμές πρέπει να ανήκουν σε ένα και μόνο συγκεκριμένο γράφημα.
Επίσης, κάθε γράφημα παρέχει πρόσβαση σε όλες τις ακμές και τους κόμβους που ανήκουν σ' αυτό, κάθε κόμβος παρέχει πρόσβαση στις ακμές που τον συνδέουν, και από κάθε ακμή μπορούμε να έχουμε πρόσβαση στους κόμβους που συνδέει, τους κόμβους-πηγές και τους κόμβους-στόχους.

4.2. Πρόσβαση στα στοιχεία του γραφήματος

Η πρόσβαση στα στοιχεία του γραφήματος, δηλαδή στις ακμές και τους κόμβους του, γίνεται μέσα από το γράφημα με τη βοήθεια διαπροσπελαστών (iterators), των οποίων η λειτουργία στηρίζεται στην προσπέλαση μιας αλυσίδας αντικειμένων. Συγκεκριμένα αυτό επιτυγχάνεται με τις διεπαφές (interfaces) EdgeIterator και NodeIterator, για τις ακμές και τους κόμβους αντίστοιχα, οι οποίες βασίζουν τη λειτουργία τους στη διεπαφή Iterator της Java.

4.3. «Αποθήκευση» των στοιχείων του γραφήματος

Για την αποθήκευση των κόμβων και των ακμών του γραφήματος, χρησιμοποιείται η έννοια της συνδεδεμένης λίστας (Linked List). Συγκεκριμένα οι κλάσεις NodeList και EdgeList, οι οποίες χρησιμοποιούνται αντίστοιχα για τη λίστα των κόμβων και τη λίστα των ακμών, συχνά είναι παράμετροι σε ή επιστρέφονται από διάφορες μεθόδους της βιβλιοθήκης.

4.4. Δημιουργία γραφημάτων, κόμβων και ακμών

Η κλάση Graph παρέχει ένα κατασκευαστή με τον οποίο δημιουργείται ένα άδειο γράφημα.

```
// Create a new, empty graph.
Graph graph = new Graph();
```

Για τη δημιουργία κόμβων και ακμών στο γράφημα αξίζει να τονίσουμε ότι μόνο η κλάση Graph παρέχει αυτή τη δυνατότητα, καθώς επίσης για τη δημιουργία ακμής απαραίτητη προϋπόθεση είναι να υπάρχουν ήδη στο γράφημα οι δύο κόμβοι που συνδέει.

```
// ‘graph’ is of type Graph
// Create 2 nodes.
Node n1 = graph.createNode();
Node n2 = graph.createNode();
// Create an edge with n1 as source node and n2 as target node
graph.createEdge(n1, n2);
```

4.5. Διαγραφή κόμβων και ακμών

Η κλάση Graph παρέχει τη δυνατότητα διαγραφής όλων των κόμβων και των ακμών με μία εντολή αλλά και διαγραφή ενός συγκεκριμένου κόμβου ή ακμής.
Σημείωση: Κατά τη διαγραφή ενός κόμβου διαγράφονται επίσης και οι ακμές που συνδέονται μ’ αυτόν.

```
// 'graph' is of type Graph
// ’n’ is of type Node
// ‘e’ is of type Edge
 // Remove all graph elements from the graph at once.
graph.clear();

// Remove a node and its connecting edges from the graph
graph.removeNode(n);

// Remove an edge
graph.removeEdge(e);
```

4.6. Περισσότερες λειτουργίες που παρέχει η κλάση Graph

Η οντότητα γράφημα εκτός από μεθόδους που επιστρέφουν το σύνολο των κόμβων και των ακμών που περιέχει, παρέχει επίσης μεθόδους για έλεγχο αν μία ακμή ή ένας κόμβος ανήκουν στο γράφημα, καθώς επίσης αν υπάρχει ακμή που συνδέει δύο συγκεκριμένους κόμβους.

```
// 'graph' is of type Graph
// ’n1’ is of type Node
// ’n2’ is of type Node
// ‘e’ is of type Edge

 // Get the number of nodes in the graph.
int nodeCount = graph.nodeCount();

// Get the number of edges in the graph.
int edgeCount = graph.edgeCount();

// Check if node ‘n1’ belongs to the graph
Boolean containsNode= graph.containsNode (n1);

// Check if edge ‘e’ belongs to the graph
boolean containsEdge= graph.containsEdge (e);

// Check if the edge that connects the nodes ‘n1’ and ‘n2’ belongs to the graph
boolean containsEdge= graph.containsEdge (n1,n2);
```

Επιπρόσθετα, κάθε οντότητα γράφημα προσφέρει πρόσβαση στα στοιχεία του γραφήματος, δηλαδή στους κόμβους και τις ακμές του, με ποικίλους τρόπους. Είτε επιστρέφει τους διαπροσπελαστές NodeIterator και EdgeIterator αντιστοίχως για το σύνολο των κόμβων και το σύνολο των ακμών του γραφήματος, ή δημιουργεί διανύσματα που περιέχουν όλους τους κόμβους ή όλες τις ακμές του γραφήματος. Για επιπλέον ευκολία υπάρχουν επίσης μέθοδοι που επιστρέφουν το πρώτο και το τελευταίο στοιχείο της κάθε λίστας κόμβων ή ακμών.

Σημείωση: Με τους διαπροσπελαστές παρέχεται απλή προσπέλαση των στοιχείων του γραφήματος, ενώ με τα διανύσματα υπάρχει η δυνατότητα τροποποίησης, όπως για παράδειγμα αφαίρεση κάποιων κόμβων ή αλλαγή της θέσης τους στο διάνυσμα, χωρίς όμως να επηρεάζεται το σύνολο των κόμβων στο γράφημα επειδή τα διανύσματα αποτελούν αντίγραφα των εν λόγω στοιχείων του γραφήματος.

```
// ‘graph’ is of type Graph

// Get the first and the last node of the graph.
Node firstNode = graph.firstNode ();
Node lastNode = graph.lastNode();

// Get the first and the last edge of the graph.
Edge firstEdge = graph.firstEdge();
Edge lastEdge = graph.lastEdge ();

// Get an Iterator to iterate over all  nodes of the node set from the graph.
NodeIterator nodeIter= graph.nodes ();

// Get an Iterator of all the edges over all edges of the edge set from the graph.
EdgeIterator edgeIter= graph.edges ();

// Get an array of all the nodes.
Object nodes [ ] = graph.getNodeArray ();

// Get an array of all the edges.
Object edges [ ] = graph.getEdgeArray ();
```

4.7. Στοιχεία του γραφήματος

Όπως αναφέρθηκε και πιο πάνω, κάθε κόμβος ή ακμή συνδέεται με ένα και μόνο γράφημα στο οποίο και ανήκει.

```
// ’node’ is of type Node
 // Get the graph the node belongs to.
Graph graph=node.getGraph();
```

Επίσης, κάθε στοιχείο του γραφήματος γνωρίζει τη θέση του στο αντίστοιχο σύνολο στοιχείων του γραφήματος στο οποίο ανήκει.

```
// ’node’ is of type Node
// ‘edge’ is of type Edge
 // Get the node’s position.
int nodeIndex =node.getIndex();

// Get the edge’s position.
int edgeIndex =edge.getIndex();
```

4.7.1. Κλάση Node

Η κλάση Node είναι υπεύθυνη για οτιδήποτε σχετίζεται με τις ακμές που συνδέουν ένα κόμβο. Συγκεκριμένα, ένα αντικείμενο τύπου Node γνωρίζει το συνολικό αριθμό των ακμών που συνδέονται μ’ αυτόν, τον αποκαλούμενο βαθμό του κόμβου (degree), καθώς επίσης περιέχει μεθόδους με τις οποίες μπορεί να εξεταστεί αν υπάρχει ακμή που τον συνδέει με έναν άλλο δοθέντα κόμβο.
Επίσης, από ένα κόμβο μπορούμε να έχουμε πρόσβαση σε μία λίστα με τους γειτονικούς σε αυτόν κόμβους, αλλά και στις παρακείμενες ακμές του.

```
// 'node' is of type Node
// Get the number of edges at a node.
int degree = node.degree();

Node firstNode = graph.firstNode();
Node lastNode = graph.lastNode();
// Check whether there is an edge connecting the first and the last node of the
graph.
Edge e = firstNode.getEdge(lastNode);
//if such an edge doesn't exist e=null

// Get a list of all the neighbors of a node
NodeList neighbors= node.getNeighbors();
// Get a list of all the adjacency edges of a node
EdgeList edges= node.getAdjacencyEdges();
```

4.7.2. Κλάση Edge

Οι πιο σημαντικές πληροφορίες που διατηρεί μια οντότητα Edge είναι ο κόμβος-πηγή και ο κόμβος-στόχος. Επίσης παρέχει και μέθοδο η οποία επιστρέφει τον κόμβο στην αντίθετη πλευρά όταν είναι γνωστός ο ένας από τους δύο κόμβους τους οποίους συνδέει.

```
// 'edge' is of type Edge.
// Get the two end nodes of an edge.
Node source = edge.getSource();
Node target = edge.getTarget();

// Return the node at this edge which is on the opposite side of the given node
Node opposite = edge.getOppositeNode(source);
```

4.8. Μηχανισμοί προσπέλασης

Ένας διαπροσπελαστής προσφέρει πρόσβαση στο σύνολο των αντικειμένων πάνω στο οποίο τρέχει, γνωρίζει αν υπάρχει επόμενο αντικείμενο στο σύνολο αυτό, καθώς επίσης μετακινεί το δρομέα στο επόμενο αντικείμενο της λίστας.

```
// ‘graph’ is of type Graph
// Forward iterate over all nodes of the node set from the graph.
for (NodeIterator nodeIter = graph.nodes (); nodeIter.hasNextNode(); ) {

Node currentNode =nodeIter.nextNode();

 }

// Forward iterate over all edges of the edge set from the graph.
for (EdgeIterator edgeIter = graph.edges (); edgeIter.hasNextEdge(); ) {

        Edge currentEdge =edgeIter.nextEdge();
}
```

Εικόνα 2 Παράδειγμα προσπέλασης των στοιχείων του γραφήματος με τη χρήση διαπροσπελαστών.

4.9. Χρήση HashMap για δέσμευση πληροφορίας στα στοιχεία του γραφήματος

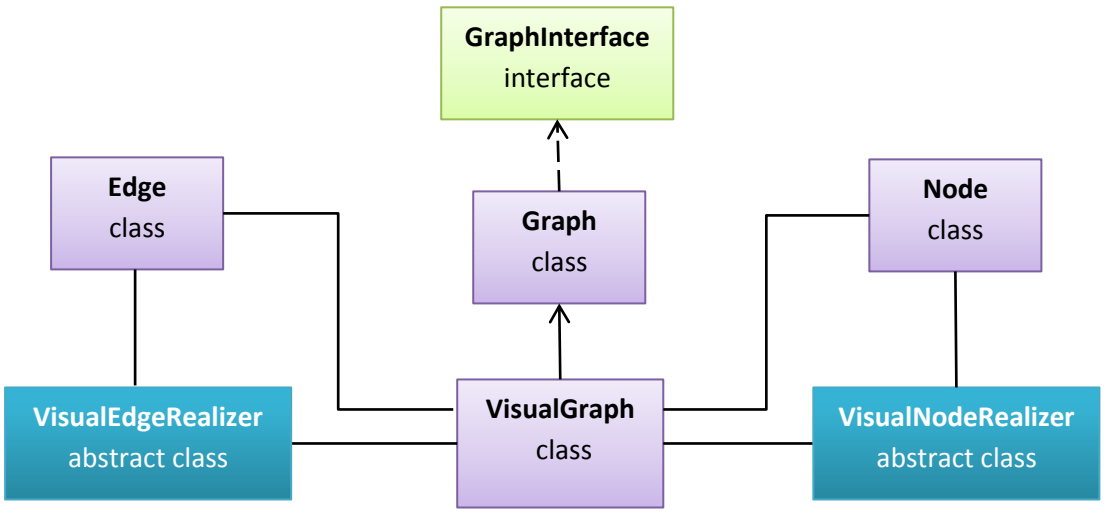
Για τη διαδικασία αυτή, η βιβλιοθήκη περιέχει τις διεπαφές NodeMap και EdgeMap για τους κόμβους και τις ακμές αντίστοιχα. Οι χάρτες όπως μπορούμε να τους αποκαλέσουμε, καλύπτουν όλα τα στοιχεία του συνόλου είτε των ακμών ή των κόμβων. Για παράδειγμα, ένας NodeMap παρέχει τιμές για όλους τους κόμβους του γραφήματος.
Στο πιο κάτω παράδειγμα φαίνονται προκαθορισμένες υλοποιήσεις των διεπαφών αυτών. Οι χάρτες που επιστρέφονται από αυτές τις μεθόδους, διατηρούν ακριβώς μία τιμή για κάθε δοθέν κλειδί, ανεξάρτητα με το πόσες φορές θα κληθούν οι «set» μέθοδοι.

```
// ‘graph’ is of type Graph
// Obtain a new NodeMap default implementation from the graph.
NodeMap nodeMap=graph.createNodeMap();

// Set values for some of the nodes
nodeMap.setDouble(graph.firstNode(), 7.7);

nodeMap.setDouble(graph.lastNode(), 13.3);
```

4.10. Απεικόνιση και Επεξεργασία των γραφημάτων

Το πακέτο Visual περιέχει κλάσεις για τις λειτουργίες της Διεπαφής Χρήστη (User Interface).
Στο πακέτο αυτό η υλοποίηση της οντότητας γράφημα παρέχεται με την κλάση VisualGraph η οποία κληρονομεί την κλάση Graph, και ουσιαστικά ενισχύει τη δομή του γραφήματος με πληροφορίες για την οπτικοποίηση των στοιχείων του.
Οι αφηρημένες κλάσεις VisualNodeRealizer και VisualEdgeRealizer στο πακέτο αυτό, είναι υπεύθυνες για να οπτικοποιήσουν τους κόμβους και τις ακμές του γραφήματος, αντίστοιχα. Ο ρόλος των κλάσεων αυτών είναι πολύ σημαντικός, αφού κατέχουν και διαχειρίζονται τις πληροφορίες για την κατάσταση των στοιχείων του γραφήματος, και είναι ουσιαστικά υπεύθυνες για να διασφαλίσουν την εύρυθμη αλληλεπίδραση του χρήστη με το γράφημα.

4.11. Κλάσεις VisualGraph, VisualNodeRealizer, και VisualEdgeRealizer



Κάθε φορά που προστίθεται ένα νέο στοιχείο στο γράφημα, είναι απαραίτητο να καθοριστεί ο «realizer» του. Αυτό μπορεί να επιτευχθεί είτε αυτόματα με την προκαθορισμένη τιμή του «realizer», ή εναλλακτικά σε κάθε στοιχείο μπορεί να συσχετιστεί ένας εξατομικευμένος realizer.

Σημείωση: Ως προκαθορισμένοι realizers για τους κόμβους και τις ακμές χρησιμοποιούνται οι RectNodeVisualizer και StraightEdgeVisualizer, αντίστοιχα.

```
// Getter and Setter methods, of the class 'VisualGraph', for the realizers

VisualNodeRealizer getNodeVisualizer (Node n);
VisualEdgeRealizer getEdgeVisualizer(Edge e);

void setNodeVisualizer(Node n, VisualNodeRealizer nodeVisualizer);
void setEdgeVisualizer(Edge e, VisualEdgeRealizer edgeVisualizer);
```

Εικόνα 3 Παράδειγμα των μεθόδων που συσχετίζουν την κλάση VisualGraph με τους Realizers.

4.11.1. Δημιουργία στοιχείων με προκαθορισμένες τιμές για τους realizers

Οι κόμβοι και οι ακμές που δημιουργούνται από τις μεθόδους που παρουσιάζονται στο πιο κάτω απόσπασμα κώδικα, έχουν προκαθορισμένη τιμή realizer, δηλαδή RectNodeRealizer για τους κόμβους και StraightEdgeRealizer για τις ακμές.

```
// Node creation resorting to RectNodeVisualizer, which is the default realizer, binding
// Both methods are from VisualGraph.
Node createNode ();
Node createNode (String id);

// Edge creation resorting to StraightEdgeVisualizer, which is the default realizer, binding
// The method is from VisualGraph.
Edge createEdge (Node source, Node target);
```

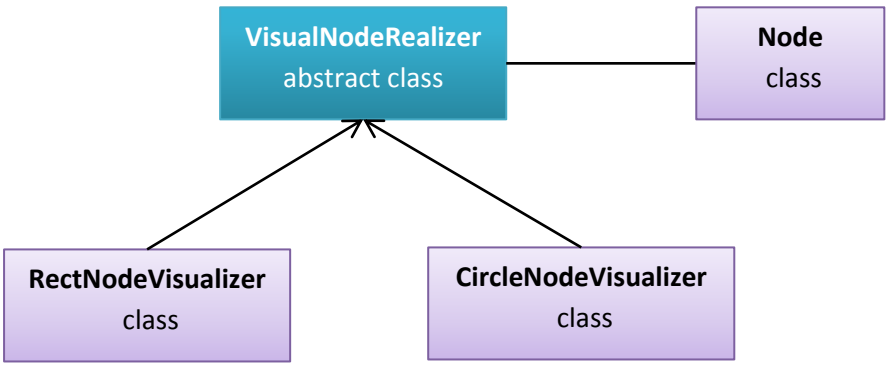
4.11.2. Δημιουργία στοιχείων με δοθείσες τιμές για τους realizers

Εκτός από τις set μεθόδους για τους realizers που αναφέρθηκαν πιο πάνω, ένα στοιχείο του γραφήματος μπορεί να συσχετισθεί κατά τη δημιουργία του με ένα realizer εκτός του προκαθορισμένου.

```
// All the methods mentioned are from VisualGraph.

// Node creation with explicit realizer binding
Node createNode (VisualNodeRealizer nodeVisualizer);
Node createNode (VisualNodeRealizer nodeVisualizer , String id);

// Edge creation with explicit realizer binding
Edge createEdge (Node source, Node target, VisualEdgeRealizer edgeVisualizer);
```

4.11.3. Node Realizers

Οι Node realizers όπως αναφέρθηκε και πιο πάνω, είναι υπεύθυνοι για να συσχετίσουν γραφικά τους κόμβους του γραφήματος σε ένα view.
Η αφηρημένη κλάση VisualNodeRealizer παρέχει τη βάση για τις διάφορες υλοποιήσεις που μπορούν να δημιουργηθούν.
Στο πακέτο Visual παρέχονται δύο τέτοιες υλοποιήσεις, οι οποίες είναι οι κλάσεις ***RectNodeVisualizer*** και ***CircleNodeVisualizer***, για κόμβους σε σχήμα τετραγώνου και κύκλου αντίστοιχα.



Εικόνα 4 Ιεραρχία των Node Realizers.

Οι δύο αυτές κλάσεις, καθώς και οποιαδήποτε άλλη υλοποίηση της VisualNodeRealizer, θα πρέπει να παρέχουν κώδικα για τις αφηρημένες μεθόδους, οι οποίες διαφέρουν ανάλογα με τη μορφή αναπαράστασης του κόμβου που υποστηρίζει η κάθε υλοποίηση.

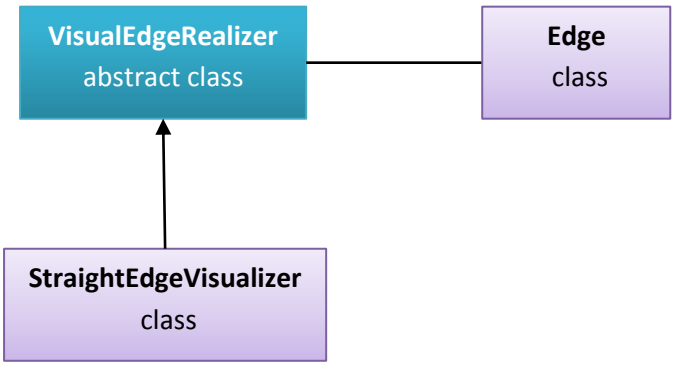
```
abstract void draw (Node node, Canvas canvas);
```

Εικόνα 5 Αφηρημένη μέθοδος της κλάσης VisualNodeRealizer.

Επίσης, η κλάση VisualNodeRealizer περιέχει get και set μεθόδους για τις άνω αριστερά συντεταγμένες x και y του κόμβου (ή του πλαισίου του(bounding box) στην περίπτωση του κύκλου), τις συντεταγμένες του κέντρου του, το χρώμα του, το ύψος και το πλάτος του, καθώς και άλλα χαρακτηριστικά που αφορούν την αναπαράστασή του.
Ενδεικτικά φαίνονται πιο κάτω, μερικές από αυτές τις μεθόδους.

```
// Getter and Setter methods, of the class 'VisualNodeRealizer'

float getCenterX();
float getHeight ();
int getFillColor ();

void setCenterX(float centerX);
void setHeight(float height);
void setFillColor(int color);
```

4.11.4. Edge Realizers



Εικόνα 6 Ιεραρχία των Edge Realizers.

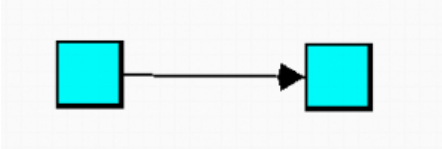
Αντίστοιχα με τους Node Realizers, οι Edge Realizers είναι υπεύθυνοι για να συσχετίσουν γραφικά τις ακμές του γραφήματος σε ένα view, και η αφηρημένη κλάση VisualEdgeRealizer παρέχει τη βάση για τις διάφορες υλοποιήσεις.
Στο πακέτο visual παρέχεται η υλοποίηση ***StraightEdgeVisualizer,*** η οποία αναπαριστά τις ακμές του γραφήματος ως απλές, ευθείες γραμμές.
Η κλάση StraightEdgeVisualizer καθώς και οποιαδήποτε άλλη υλοποίηση της VisualEdgeRealizer, θα πρέπει να παρέχει κώδικα για τις αφηρημένες μεθόδους, οι οποίες διαφέρουν ανάλογα με τη μορφή αναπαράστασης της ακμής που υποστηρίζει η κάθε υλοποίηση.

```
abstract void draw (Edge edge, Canvas canvas);
abstract boolean contains (float x, float y);
abstract void setBoundingBox (Edge edge);
```

Εικόνα 7 Αφηρημένες μέθοδοι της κλάσης VisualEdgeRealizer.

Επίσης, η κλάση VisualEdgeRealizer περιέχει get και set μεθόδους για το χρώμα της ακμής και το είδος της, αν θα είναι δηλαδή απλή ή διακεκομμένη (dashed) γραμμή.

```
// Getter and Setter methods, of the class 'VisualEdgeRealizer'

int getLineColor();
byte getLineStyle ();

void setLineColor(int lineColor);
void setLineStyle(byte lineStyle);
```



Εικόνα 8 Αναπαράσταση ακμής με την κλάση StraightEdgeVisualizer.

4.11.4.1. Κλάση StraightEdgeVisualizer

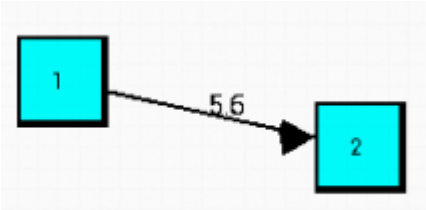
Η κλάση StraightEdgeVisualizer εκτός από τις μεθόδους τις οποίες κληρονομεί από την αφηρημένη κλάση VisualEdgeRealizer, περιέχει επίσης τις private μεθόδους:

- getInterscectionSourceX(Edge e)
- getInterscectionSourceY(Edge e)
- getInterscectionTargetX(Edge e)
- getInterscectionTargetY (Edge e)
- getXOfEdge(Edge e, float y)
- getYOfEdge(Edge e, float x)
- pointOfInterception(PointF start1,PointF end1, PointF start2, PointF end2)

οι οποίες χρησιμοποιούνται για να υπολογιστούν τα σημεία τομής της κάθε ακμής με τους δύο κόμβους τους οποίους συνδέει, πληροφορίες οι οποίες χρειάζονται για τον υπολογισμό του «bounding box» της ακμής.

4.12. Κλάση Label - Χρήση Ετικετών Κειμένου και Realizers

Πληροφορίες κειμένου που αφορούν τους κόμβους και τις ακμές, όπως είναι για παράδειγμα τα βάρη στις ακμές, μπορούν να αναπαρασταθούν ως ετικέτες κειμένου με τις αντίστοιχες μεθόδους στους realizers των κόμβων και των ακμών. Το κείμενο αναγράφεται για τους κόμβους στο κέντρο του κόμβου και για τις ακμές στο μέσο της ακμής, όπως φαίνεται στο πιο κάτω σχήμα.




```
// nodeRealizer is of type 'VisualNodeRealizer'

Label nodeLabel= new Label (String text);
nodeRealizer.setDisplayLabel(boolean displayLabel);
nodeRealizer.setLabel(nodeLabel);

// edgeRealizer is of type 'VisualEdgeRealizer'

Label edgeLabel= new Label (String text);
edgeRealizer.setLabel(edgeLabel);
```

Εικόνα 9 Δημιουργία ετικετών κειμένου σε κόμβους και ακμές.

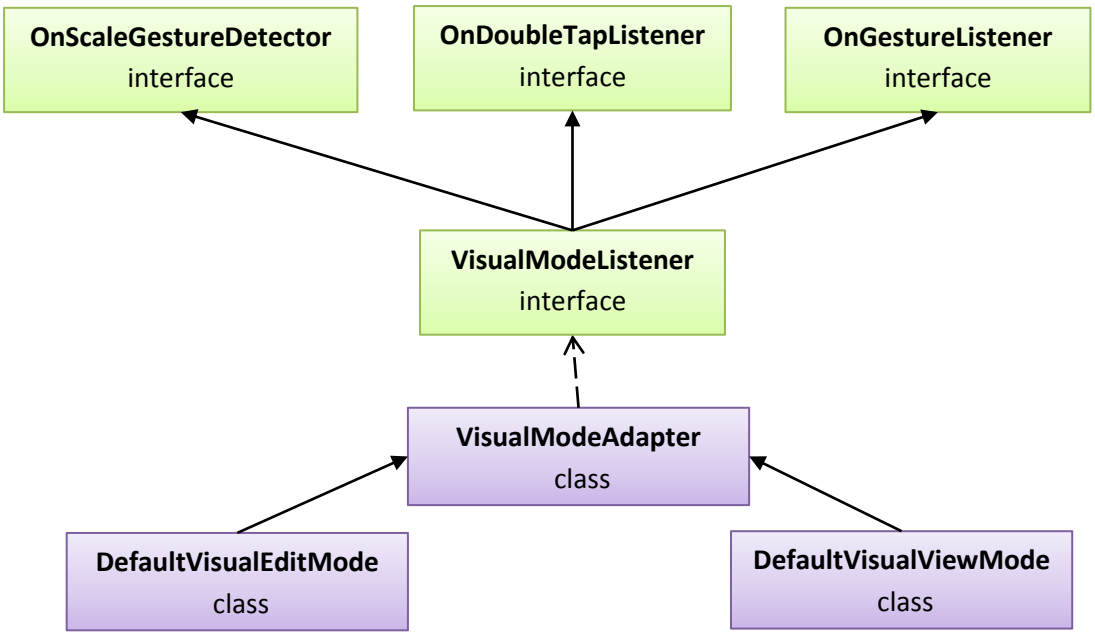
4.13. Αλληλεπίδραση με τον χρήστη και Listeners

Η κλάση VisualModeAdapter υλοποιεί τη διεπαφή VisualModeListener χρησιμοποιώντας μεθόδους με άδεια σώματα. Από τη VisualModeListener προκύπτουν οι κλάσεις ***DefaultVisualEditMode*** και ***DefaultVisualViewMode,*** που αποτελούν προκαθορισμένες υλοποιήσεις του VisualModeListener μέσω επέκτασης της κλάσης VisualModeAdapter.
Οι κλάσεις αυτές είναι υπεύθυνες για τη διαχείριση των ενεργειών του χρήστη πάνω στην οθόνη, όπως είναι για παράδειγμα το απλό και διπλό άγγιγμα, το παρατεταμένο άγγιγμα, η μεγέθυνση χρησιμοποιώντας δύο δάκτυλα (pinch zoom), η κύλιση (panning), αλλά και άλλες ενέργειες όπως η ένωση δύο κόμβων ή η μετακίνηση ενός κόμβου.
Η διεπαφή ***VisualModeListener,*** την οποία υλοποιεί η κλάση VisualModeAdapter,

επεκτείνει τρεις διεπαφές του Android API οι οποίες είναι υπεύθυνες για την ανίχνευση των διαφόρων ενεργειών του χρήστη στη οθόνη. Πιο κάτω αναφέρονται οι διεπαφές αυτές με τις μεθόδους που περιέχουν.

- ***OnGestureListener***
  - onDown(MotionEvent e)
  - onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY)
  - onLongPress(MotionEvent e)
  - onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY)
  - onShowPress(MotionEvent e)
  - onSingleTapUp(MotionEvent e)
- ***OnDoubleTapListener***
  - onDoubleTap(MotionEvent e)
  - onDoubleTapEvent(MotionEvent e)
  - onSingleTapConfirmed(MotionEvent e)
- ***OnScaleGestureListener***
  - onScale(ScaleGestureDetector detector)
  - onScaleBegin(ScaleGestureDetector detector)
  - onScaleEnd (ScaleGestureDetector detector)

- *Σημείωση: Όταν χρησιμοποιούνται παράλληλα ο onGestureListener και ο onDoubleTapListener για την ανίχνευση του απλού αγγίγματος (single tap), πρέπει να χρησιμοποιείται η μέθοδος onSingleTapConfirmed του OnDoubleTapListener και όχι η onSingleTapUp.*



Εικόνα 10 Ιεραρχία των Listeners

Μέθοδοι της κλάσης VisualModeListener:

- onDown(MotionEvent e)
- onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY)
- onLongPress(MotionEvent e)
- onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY)
- onShowPress(MotionEvent e)
- onSingleTapUp(MotionEvent e)
- onDoubleTap(MotionEvent e)
- onDoubleTapEvent(MotionEvent e)
- onSingleTapConfirmed(MotionEvent e)
- onScaleBegin(ScaleGestureDetector detector)
- onScaleEnd (ScaleGestureDetector detector)
- connectNodes(float sourceX,float sourceY,float targetX,float targetY)
- moveNode(float sourceX,float sourceY,float targetX,float targetY)
- getGestureDetector()
- getScaleDetector()

Η διεπαφή VisualModeListener εκτός από τις μεθόδους τις οποίες κληρονομεί από τις τρεις διεπαφές του Android, παρέχει επίσης τις μεθόδους connectNodes, moveNode, getGestureDetector() και getScaleDetector().
Η connectNodes καλείται όταν στο σημείο που ακουμπά ο χρήστης στην οθόνη αλλά και στο σημείο στο οποίο ανεβάζει το χέρι του, υπάρχει κάποιος κόμβος του γραφήματος. Η moveNode καλείται όταν στο σημείο που ακουμπά ο χρήστης στην οθόνη υπάρχει κόμβος, αλλά δεν υπάρχει κόμβος στο σημείο στο οποίο ανεβάζει το χέρι του. Οι άλλες δύο μέθοδοι, getGestureDetector() και getScaleDetector(), επιστρέφουν τα αντικείμενα τύπου GestureDetector και ScaleGestureDetector που

αντιστοιχούν σε κάθε Listener.

4.13.1. Κλάση DefaultVisualEditMode

Η κλάση DefaultVisualEditMode παρέχει υλοποιήσεις των διαφόρων ενεργειών του χρήστη στην οθόνη οι οποίες αφορούν την τροποποίηση των στοιχείων του γραφήματος, όπως για παράδειγμα τη δημιουργία ενός καινούριου κόμβου ή ακμής, την αλλαγή του μεγέθους ενός κόμβου ή τη μετακίνησή του. Επίσης, παρέχει υλοποίηση για το παρατεταμένο πάτημα του χρήστη σε ένα κόμβο η ακμή, σύμφωνα με την οποία εμφανίζεται ένα παράθυρο διαλόγου (AlertDialog) στο οποίο υπάρχουν επιλογές για τροποποιήσεις των κόμβων και των ακμών αντίστοιχα, όπως για παράδειγμα αλλαγή χρώματος, σχήματος, προσθήκη ετικέτας κειμένου και διαγραφή.

Σημείωση: Οι διάλογοι οι οποίοι εμφανίζονται κατά το παρατεταμένο πάτημα σε ένα κόμβο ή ακμή δεν μπορούν να τροποποιηθούν έτσι ώστε να παρέχουν περισσότερες ή λιγότερες επιλογές. Αν κάποιος επιθυμεί διαφορετικές επιλογές θα πρέπει να δημιουργήσει εξατομικευμένη υλοποιήση της onLongPress μεθόδου σε έναν άλλο VisualModeListener.

Πιο κάτω φαίνονται μερικά αποσπάσματα του κώδικα της κλάσης DefaultVisualEditMode για τη δημιουργία κόμβου και ακμής στο γράφημα, και για τη μετακίνηση ενός κόμβου.

```
//      method “onSingleTapConfirmed” from class “DefaultVisualEdiMode”
//      view is the VisualGraphView on which the graph is related
public boolean onSingleTapConfirmed(MotionEvent event){
          …
//        if nothing was selected create a node
          if (!somethingSelected){
              float objectNewX = event.getX();
              float objectNewY = event.getY();
              Node v = view.getEmbeddedGraph().createNode();
              float width=view.getEmbeddedGraph().getNodeVisualizer(v).getWidth();
              float height=view.getEmbeddedGraph().getNodeVisualizer(v).getHeight();
              view.getEmbeddedGraph().getNodeVisualizer(v).setCenterX(objectNewX);
              view.getEmbeddedGraph().getNodeVisualizer(v).setCenterY(objectNewY);
              view.getEmbeddedGraph().getNodeVisualizer(v).setX(objectNewX-(width/2));
              view.getEmbeddedGraph().getNodeVisualizer(v).setY(objectNewY-(height/2));
              view.getEmbeddedGraph().getNodeVisualizer(v).setBoundingBox();
          }
          …
view.invalidate();
return true; } }
```

Εικόνα 11 Απόσπασμα κώδικα για τη δημιουργία κόμβου

```
//      method “connectNodes”  from class “DefaultVisualEdiMode”
//      view is the VisualGraphView on which the graph is related
public void connectNodes(float sourceX,float sourceY,float targetX,float targetY){
Node source=this.nodeContaining(sourceX,sourceY);
Node target=this.nodeContaining(targetX, targetY);
            …
//                      ***EDGE CREATION***
Edge edge = view.getEmbeddedGraph().createEdge(source,target);
float sourceCenterX=view.getEmbeddedGraph().getNodeVisualizer(source).getCenterX();
float sourceCenterY=view.getEmbeddedGraph().getNodeVisualizer(source).getCenterY();
float targetCenterX=view.getEmbeddedGraph().getNodeVisualizer(target).getCenterX();
float targetCenterY=view.getEmbeddedGraph().getNodeVisualizer(target).getCenterY();
view.getEmbeddedGraph().getEdgeVisualizer(edge).setSourcePoint(new PointF(sourceCenterX,
sourceCenterY));
view.getEmbeddedGraph().getEdgeVisualizer(edge).setTargetPoint(new PointF(targetCenterX,
targetCenterY));
view.getEmbeddedGraph().getEdgeVisualizer(edge).setBoundingBox(edge);
            …
            }
```

Εικόνα 12 Απόσπασμα κώδικα για τη δημιουργία ακμής

```
public void moveNode(float sourceX,float sourceY,float targetX,float targetY){
//                      ***NODE MOVE***
Node node=this.nodeContaining(sourceX,sourceY);
view.getEmbeddedGraph().getNodeVisualizer(node).setCenterX(targetX);
view.getEmbeddedGraph().getNodeVisualizer(node).setCenterY(targetY);
float width=view.getEmbeddedGraph().getNodeVisualizer(node).getWidth();
float height=view.getEmbeddedGraph().getNodeVisualizer(node).getHeight();
view.getEmbeddedGraph().getNodeVisualizer(node).setX(targetX-(width/2));
view.getEmbeddedGraph().getNodeVisualizer(node).setY(targetY-(height/2));
view.getEmbeddedGraph().getNodeVisualizer(node).setBoundingBox();
for (NodeIterator nodeIterator=view.getEmbeddedGraph().nodes();
nodeIterator.hasNextNode();){
 Node currentNode=nodeIterator.nextNode();
 if(view.getEmbeddedGraph().getEdgeVisualizer(node.getEdge(currentNode))!=null){
            view.getEmbeddedGraph().getEdgeVisualizer(node.getEdge(currentNode))
            .setBoundingBox(node.getEdge(currentNode));
                        }
            }
}
```

Εικόνα 13 Απόσπασμα κώδικα για τη μετακίνηση κόμβου

4.13.2. Κλάση DefaultVisualViewMode

Η κλάση DefaultVisualViewMode παρέχει υλοποίηση των μεθόδων onScale και onDoubleTapEvent, με τις οποίες επιτυγχάνονται αντίστοιχα η μεγέθυνση του canvas και η επαναφορά της ορατής οθόνης στην αρχή των αξόνων, δηλαδή στην αρχική άνω αριστερή γωνία που αντιστοιχεί με το (0,0).

```
//view is of type 'VisualGraphView'
onDoubleTapEvent(MotionEvent event){
// both methods mentioned are from VisualGraphView
            view.setTranslateX(0);
            view.setTranslateY(0);
            return true;
}
onScale(ScaleGestureDetector detector){
//all methods mentioned are from VisualGraphView
             view.setScalingFactor(view.getScalingFactor()*detector.getScaleFactor());
            //Don't let the object get too small or too large.
            view.setScalingFactor(Math.max(0.5f, Math.min(view.getScalingFactor(), 5.0f)));
            view.setPivotPointX(detector.getFocusX());
            view.setPivotPointY(detector.getFocusY());
            return true; }
```

Εικόνα 14 Μέθοδοι κλάσης DefaultVisualViewMode

Σημείωση: Θα πρέπει να αναφερθεί ότι σε κάθε VisualModeListener αντιστοιχεί ένα αντικείμενο ScaleGestureDetector, ένα αντικείμενο GestureDetector, καθώς και το view το οποίο θα χειριστεί τις ενέργειες. Έτσι, στον κατασκευαστή κάθε κλάσης η οποία επεκτείνει την κλάση VisualModeAdapter θα πρέπει να δημιουργείται ένα αντικείμενο ScaleGestureDetector, ένα αντικείμενο GestureDetector, καθώς και να ορίζεται το view. Επίσης θα πρέπει να ορίζονται οι μέθοδοι getScaleDetector() και getGestureDetector() της VisualModeListener.

4.14. Κλάση VisualGraphView

Η κλάση VisualGraphView στο πακέτο Visual, η οποία επεκτείνει την κλάση View του Android API, καταλαμβάνει μία ορθογώνια περιοχή στην οθόνη και είναι υπεύθυνη για το σχεδιασμό αντικειμένων στην οθόνη, καθώς και για την ανίχνευση των διαφόρων ενεργειών του χρήστη στην οθόνη. Συγκεκριμένα, η μέθοδος onTouchEvent(MotionEvent event) της κλάσης αυτής, είναι υπεύθυνη για την ανίχνευση των διαφόρων ενεργειών του VisualModeListener.
Επίσης, κάθε VisualGraphView διατηρεί μία λίστα με τους VisualModeListener οι οποίοι ενεργούν σ' αυτό το View.
Σημείωση: Είναι απαραίτητη η κλήση της μεθόδου onTouchEvent(MotionEvent

event) στα αντικείμενα GestureDetectorκαι ScaleGestureDetector που αντιστοιχούν σε κάθε VisualModeListener της λίστας αυτής.

```
// method onTouchEvent(MotionEvent event)
//visualModes is a linkedList of VisualModeListeners
…
for (Iterator<VisualModeListener> it=this.visualModes.iterator(); it.hasNext();){
        VisualModeListener currentMode=(VisualModeListener) it.next();
        currentMode.getGestureDetector().onTouchEvent(event);
        currentMode.getScaleDetector().onTouchEvent(event);
…
```

Εικόνα 15 Κλήση της onTouchEvent στους GestureDetector και ScaleGestureDetector.

Μια οντότητα VisualGraphView μπορεί να περιέχει και να αναπαριστά μόνο ένα γράφημα, το οποίο παρέχεται με την κατάλληλη "get" μέθοδο της κλάσης. Επίσης, ο σχεδιασμός του γραφήματος επιτυγχάνεται με τη μέθοδο onDraw(Canvas canvas), η οποία σχεδιάζει σε ένα αντικείμενο canvas και καλείται κάθε φορά που το γράφημα τροποποιείται.
Στην κλάση αυτή περιέχονται επίσης μέθοδοι με τις οποίες καθορίζεται η τιμή για το συντελεστή μεγέθυνσης καθώς και για το σημείο γύρω από το οποίο θα γίνεται η μεγέθυνση, αλλά και οι τιμές για τον άξονα x και τον άξονα y σύμφωνα με τις οποίες πρέπει να μεταφράζεται το canvas για να υπάρχει συνέπεια στις συντεταγμένες ανάλογα με την κύλιση.

```
// Getter and Setter methods, of the class 'VisualGraphView', for the scaling
factor and the pivot point.

float getScalingFactor ();
float getPivotPointX ();
float getPivotPointY ();


void setScalingFactor(float scaleFactor);
void setPivotPointX (float pivotX);
void setPivotPointY (float pivotY);
```

Εικόνα 16 Μέθοδοι για καθορισμό και ανάκτηση του συντελεστή μεγέθυνσης και του οδηγού-στοιχείου.

Οι listeners τύπου VisualModeListener, προστίθενται ή αντίστοιχα αφαιρούνται από ένα view με τη βοήθεια μεθόδων της κλάσης VisualGraphView.

```
// both methods mentioned are from class 'VisualGraphView'
// Methods to add and remove a view mode.
void addVisualMode (VisualModeListener mode);
void removeVisualMode (VisualModeListener mode);
```

Εικόνα 17 Μέθοδοι συσχέτισης VisualGraphView με VisualModeListener

4.14.1. Προσθήκη εξατομικευμένου VisualModeListener σε ένα View

Εκτός από τις προκαθορισμένες υλοποιήσεις DefaultVisualEditMode και DefaultVisualViewMode, η βιβλιοθήκη παρέχει τη δυνατότητα για προσθήκη εξατομικευμένης υλοποίησης της διεπαφής VisualModeListener.
Πιο κάτω φαίνεται ένα παράδειγμα στο οποίο προστίθεται σε ένα VisualGraphView η κλάση MyListener (μια υλοποίηση του VisualModeListener), σύμφωνα με την οποία δημιουργείται ένας νέος κόμβος στο γράφημα με κάθε διπλό άγγιγμα του χρήστη στην οθόνη.

```
//graphView is of type 'VisualGraphView'
VisualModeListener myMode;
myMode=new MyListener(graphView);

graphView.addVisualMode(myMode);

 private class MyListener extends visual.VisualModeAdapter{
private ScaleGestureDetector scaleDetector;
private GestureDetector detector;
private VisualGraphView view;
public MyListener(VisualGraphView view){
            this.view=view;
            this.detector=new GestureDetector(view.getContext(),this);
            this.scaleDetector=new ScaleGestureDetector(view.getContext(),this);
                        }
public GestureDetector getGestureDetector(){
            return detector;
                        }
public ScaleGestureDetector getScaleDetector(){
            return scaleDetector;
                        }
public boolean onDoubleTapEvent(MotionEvent event){
            float objectNewX = ((event.getX()-
graphView.getPivotPointX())/graphView.getScalingFactor())-
graphView.getTranslateX()+graphView.getPivotPointX();
            float objectNewY = (event.getY()-graphView.getPivotPointY())/graphView.getScalingFactor()-
graphView.getTranslateY()+graphView.getPivotPointY();

           Node v = graphView.getEmbeddedGraph().createNode();
           float width=graphView.getEmbeddedGraph().getNodeVisualizer(v).getWidth();
           float height=graphView.getEmbeddedGraph().getNodeVisualizer(v).getHeight();
           graphView.getEmbeddedGraph().getNodeVisualizer(v).setCenterX(objectNewX);
           graphView.getEmbeddedGraph().getNodeVisualizer(v).setCenterY(objectNewY);
           graphView.getEmbeddedGraph().getNodeVisualizer(v).setX(objectNewX-(width/2));
           graphView.getEmbeddedGraph().getNodeVisualizer(v).setY(objectNewY-(height/2));
           graphView.getEmbeddedGraph().getNodeVisualizer(v).setBoundingBox();
return true;
              }

}
```

4.15. Αποθήκευση και ανάκτηση ενός γραφήματος

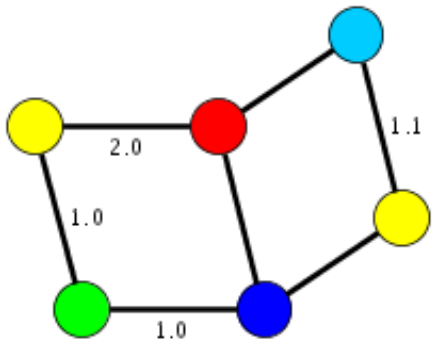
Η αποθήκευση και η ανάκτηση ενός γραφήματος επιτυγχάνεται με τις κλάσεις ***SaveToGraphML*** και ***LoadFromGraphML*** αντίστοιχα, στο πακέτο Visual.

4.15.1. GraphML αρχείο

Για αποθήκευση και ανάκτηση ενός γραφήματος, ο τύπος αρχείου που επιλέχθηκε είναι ***GraphML***, ο οποίος είναι βασισμένος σε *xml* τύπο αρχείου και χρησιμοποιείται συγκεκριμένα για γραφήματα. Περιλαμβάνει εντολές που περιγράφουν τη δομή του γραφήματος, αλλά επίσης και μηχανισμούς που επιτρέπουν την προσθήκη δεδομένων ανάλογα με τις ανάγκες κάθε εφαρμογής. Μερικά από τα κύρια χαρακτηριστικά που υποστηρίζει είναι:

- κατευθυνόμενα και μη κατευθυνόμενα γραφήματα
- υπεργραφήματα
- γραφήματα με ιεραρχική δομή
- γνωρίσματα για συγκεκριμένη εφαρμογή

Πιο κάτω φαίνεται ένα γράφημα και ο κώδικας για αυτό σε αρχείο GraphML.




```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns

http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd"
  <key id="d0" for="node" attr.name="color"
attr.type="string">
    <default>yellow</default>
  </key>
  <key id="d1" for="edge" attr.name="weight"
attr.type="double"/>
  <graph id="G" edgedefault="undirected">
    <node id="n0">
      <data key="d0">green</data>
    </node>
    <node id="n1"/>
    <node id="n2">
      <data key="d0">blue</data>
    </node>
    <node id="n3">
      <data key="d0">red</data>
    </node>
    <node id="n4"/>
    <node id="n5">
      <data key="d0">turquoise</data>
    </node>
    <edge id="e0" source="n0" target="n2">
      <data key="d1">1.0</data>
    </edge>
    <edge id="e1" source="n0" target="n1">
      <data key="d1">1.0</data>
    </edge>
    <edge id="e2" source="n1" target="n3">
      <data key="d1">2.0</data>
    </edge>
    <edge id="e3" source="n3" target="n2"/>
    <edge id="e4" source="n2" target="n4"/>
    <edge id="e5" source="n3" target="n5"/>
    <edge id="e6" source="n5" target="n4">
      <data key="d1">1.1</data>
    </edge>
  </graph>
</graphml>
```

Σε αντίθεση με πολλούς άλλους τύπους αρχείων για γραφήματα, το GraphML δε χρησιμοποιεί κάποια προκαθορισμένη δομή.

Για σκοπούς συμβατότητας, στη βιβλιοθήκη αυτή ακολουθήθηκε η δομή του GraphML όπως ορίζεται στο yEd Graph Editor.

Οι μέθοδοι των δύο κλάσεων που αναφέρθηκαν πιο πάνω, οι οποίες είναι

υπεύθυνες για την αποθήκευση και τη φόρτωση ενός γραφήματος από ένα αρχείο GraphML, είναι οι saveToGraphML και loadFromGraphML αντίστοιχα, οι οποίες είναι στατικές μέθοδοι και δέχονται ως παραμέτρους η πρώτη το γράφημα το οποίο προορίζεται για αποθήκευση και ένα Output Stream, ενώ η δεύτερη το γράφημα το οποίο θα σχεδιαστεί στην οθόνη καθώς επίσης και ένα Input Stream.

```
// static method saveToGraphML for saving a graph
static void saveToGraphML (VisualGraph graph, OutputStream output);

// static method loadFromGraphML for loading a graph
static void loadFromGraphML (VisualGraph graph, InputStream output);
```

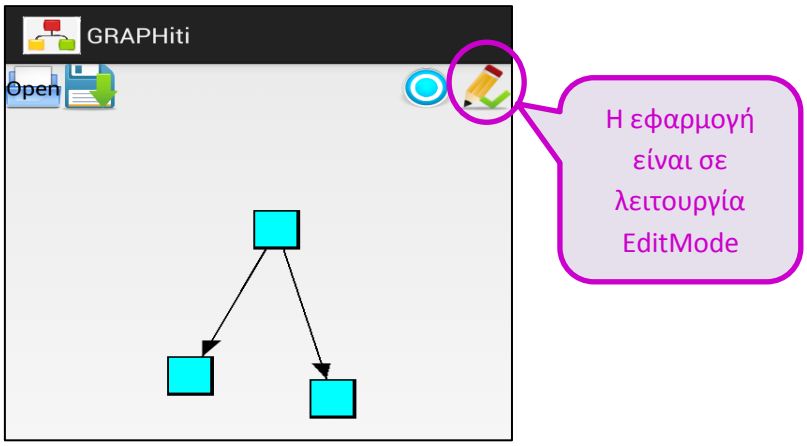
Κεφάλαιο 5: ΕΦΑΡΜΟΓΗ GRAPHiti

Πιο κάτω παρουσιάζεται η εφαρμογή «GRAPHiti» για συσκευές Android που αναπτύχθηκε με τη χρήση της βιβλιοθήκης που περιγράφηκε στο Κεφάλαιο 4.

Αξίζει να σημειωθεί ότι στην εφαρμογή αυτή χρησιμοποιήθηκαν οι προκαθορισμένες κλάσεις της βιβλιοθήκης, DefaultVisualEditMode και DefaultVisualViewMode, για την υλοποίηση της VisualModeListener.

5.1. Δημιουργία κόμβων και ακμών

Ένας κόμβος δημιουργείται με το απλό άγγιγμα του χρήστη στην οθόνη, με την προϋπόθεση ότι δεν είναι επιλεγμένο κάποιο άλλο στοιχείο του γραφήματος. Αντίστοιχα, μία ακμή δημιουργείται με το σύρσιμο του δακτύλου από ένα κόμβο σε ένα άλλο. Αν στο γράφημα υπάρχει ήδη η ακμή, τότε εμφανίζεται ένα μήνυμα ειδοποίησης, καθώς επίσης αν υπάρχει η ακμή με αντίστροφους τους κόμβους πηγή και στόχο, τότε δημιουργείται μια νέα μη-κατευθυνόμενη ακμή που συνδέει αυτούς τους δύο κόμβους.
Απαραίτητη προϋπόθεση για τη δημιουργία ενός νέου κόμβου αλλά και για τη δημιουργία μίας νέας ακμής, καθώς και για οποιαδήποτε άλλη τροποποίηση, είναι η εφαρμογή να είναι στη λειτουργία «EditMode».
Αξίζει να αναφερθεί επίσης, ότι στην περίπτωση κατά την οποία το σημείο στο οποίο αντιστοιχεί το απλό άγγιγμα του χρήστη περιέχεται σε ένα κόμβο ή σε μία ακμή του γραφήματος, τότε γίνεται επιλογή του αντίστοιχου στοιχείου, ενώ στην περίπτωση όπου υπάρχει ήδη κάποιο στοιχείο του γραφήματος επιλεγμένο και το σημείο αντιστοιχεί σε άδεια επιφάνεια τότε γίνεται αποεπιλογή του επιλεγμένου στοιχείου.



Σημείωση: Η εναλλαγή των λειτουργιών EditMode και ViewMode της εφαρμογής, επιτυγχάνεται με το πάτημα του αντίστοιχου κουμπιού το οποίο βρίσκεται στην άνω δεξιά γωνία της οθόνης. Όταν ο χρήστης θέλει για παράδειγμα να μεταφερθεί από τη λειτουργία EditMode στη λειτουργία ViewMode, κατά το πάτημα του κουμπιού

αφαιρείται από το ενεργό view ο listener «DefaultVisualEditMode» και προστίθεται ο «DefaultVisualViewMode».

```
//graphView is of type 'VisualGraphView'
//editMode is of type 'DefaultVisualEditMode'
//viewMode is of type 'DefaultVisualViewMode'
changeModesButton.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener()
{
        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
            if (isChecked) {
                graphView.addVisualMode(editMode);
                graphView.removeVisualMode(viewMode);
            }
            else {
                graphView.removeVisualMode(editMode);
                graphView.addVisualMode(viewMode);
            }
    }
    });
```

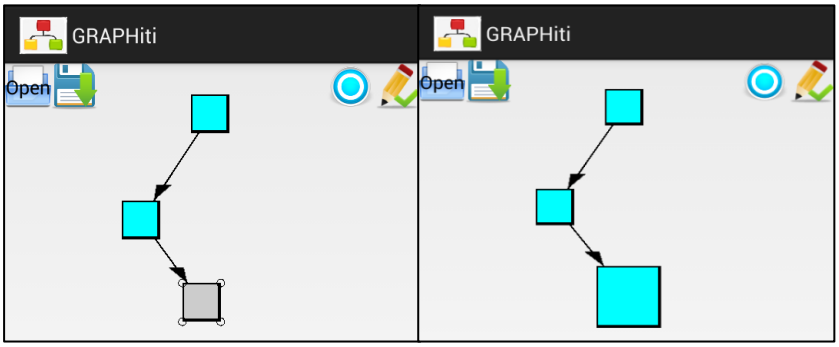
Εικόνα 18 Απόσπασμα του κώδικα της εφαρμογής για την εναλλαγή EditMode-ViewMode.

5.2. Επεξεργασία των στοιχείων του γραφήματος

Μέσα από την εφαρμογή υπάρχει η δυνατότητα τροποποίησης της οπτικής αναπαράστασης των κόμβων και των ακμών, καθώς επίσης και διαγραφή τους από το γράφημα. Συγκεκριμένα, ένας κόμβος μπορεί να μετακινηθεί, να αλλάξει μέγεθος, χρώμα και σχήμα. Αντίστοιχα, παρέχεται η δυνατότητα για αλλαγή του χρώματος μίας ακμής και η επιλογή μεταξύ κατευθυνόμενης ή μη κατευθυνόμενης ακμής.
Επιπρόσθετα, σε κάθε κόμβο ή ακμή του γραφήματος μπορεί να προστεθεί, ή να τροποποιηθεί αν υπάρχει ήδη, ετικέτα κειμένου (label).

- ***Μετακίνηση κόμβου***
  Για τη μετακίνηση ενός κόμβου, απαραίτητη προϋπόθεση είναι στο σημείο στο οποίο προορίζεται να μετακινηθεί να μην υπάρχει ήδη κάποιο στοιχείο του γραφήματος.

- ***Αλλαγή Μεγέθους του κόμβου***
  Η αλλαγή του μεγέθους ενός κόμβου επιτυγχάνεται όταν αυτός είναι επιλεγμένος, με τη βοήθεια της λειτουργίας «pinch zoom».

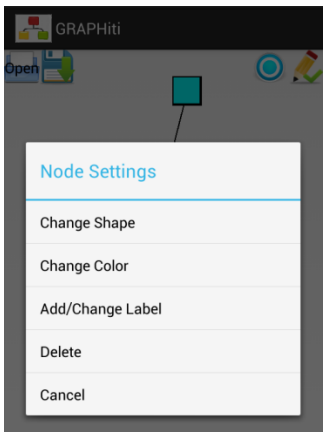


Εικόνα 19 Αλλαγή μεγέθους κόμβου.

- ***Διαγραφή και περισσότερες τροποποιήσεις των κόμβων και των ακμών***
  Με το παρατεταμένο πάτημα (Long Press) του χρήστη σε ένα κόμβο ή μία ακμή του γραφήματος, εμφανίζεται ένας διάλογος ειδοποίησης (Alert Dialog) ο οποίος παρέχει τις επιλογές τις οποίες έχει ο χρήστης για τροποποίηση των αντικειμένων.

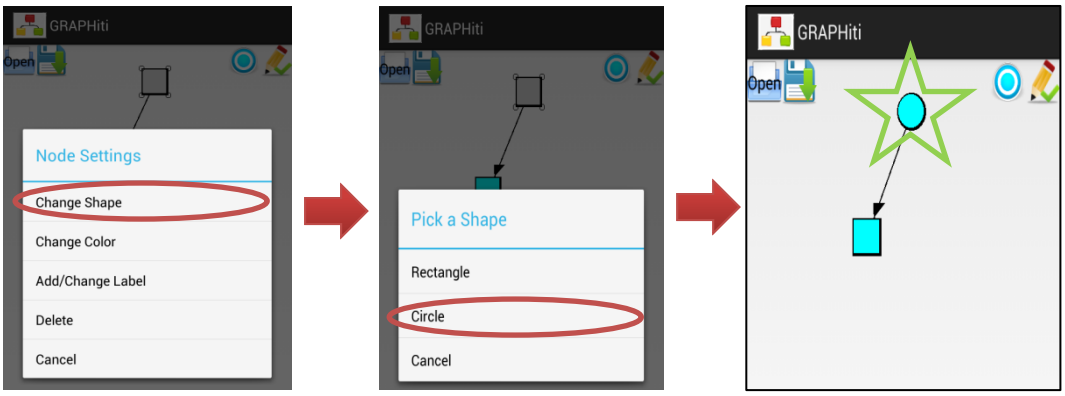
Οι επιλογές οι οποίες παρέχονται κατά το παρατεταμένο πάτημα σε ένα κόμβο, είναι:

- Αλλαγή σχήματος
- Αλλαγή χρώματος
- Προσθήκη/τροποποίηση ετικέτας κειμένου
- Διαγραφή

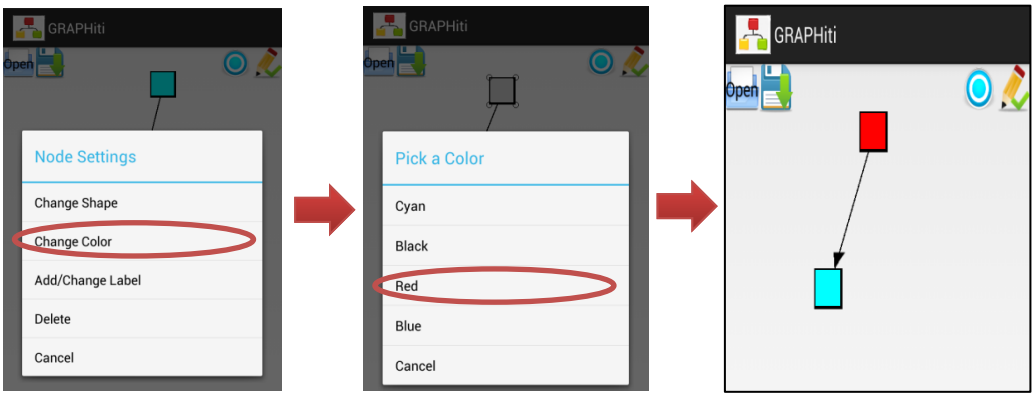


Εικόνα 20 Εμφάνιση διαλόγου ειδοποίησης κατά το long press σε ένα κόμβο.

Αλλαγή σχήματος κόμβου

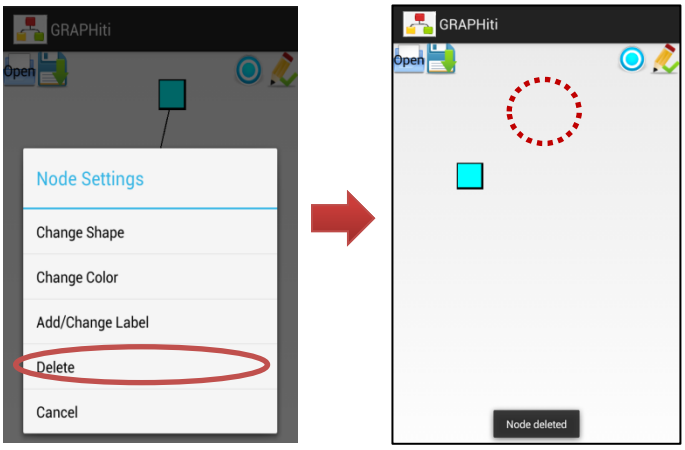


Αλλαγή χρώματος κόμβου



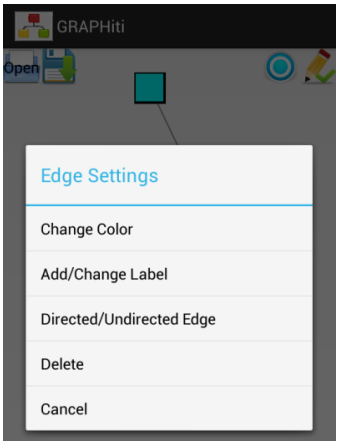
Προσθήκη Ετικέτας κειμένου σε κόμβο

Διαγραφή κόμβου



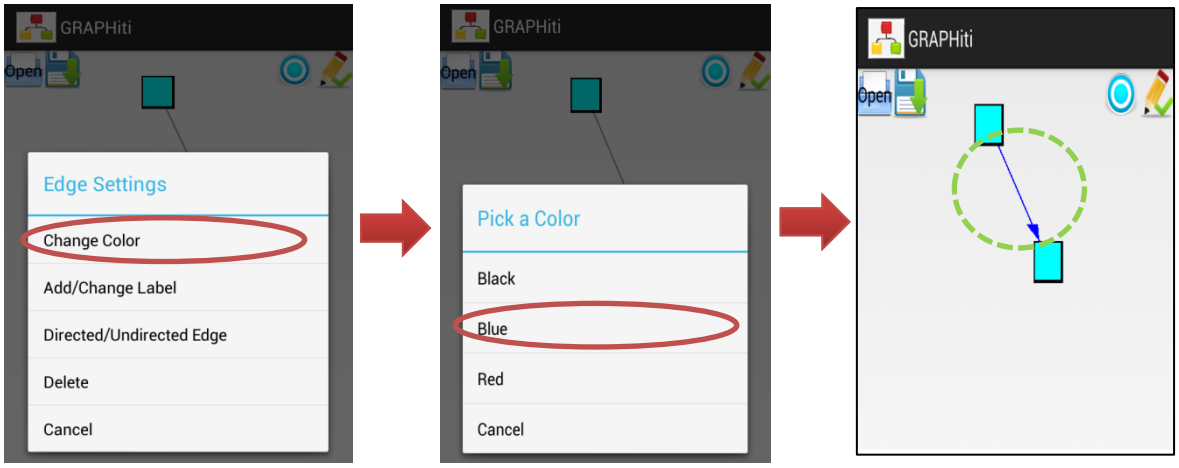
Οι επιλογές οι οποίες παρέχονται κατά το παρατεταμένο πάτημα σε μία ακμή, είναι:

- Αλλαγή χρώματος
- Προσθήκη/τροποποίηση ετικέτας κειμένου
- Επιλογή μεταξύ κατευθυνόμενης ή μη κατευθυνόμενης ακμής
- Διαγραφή



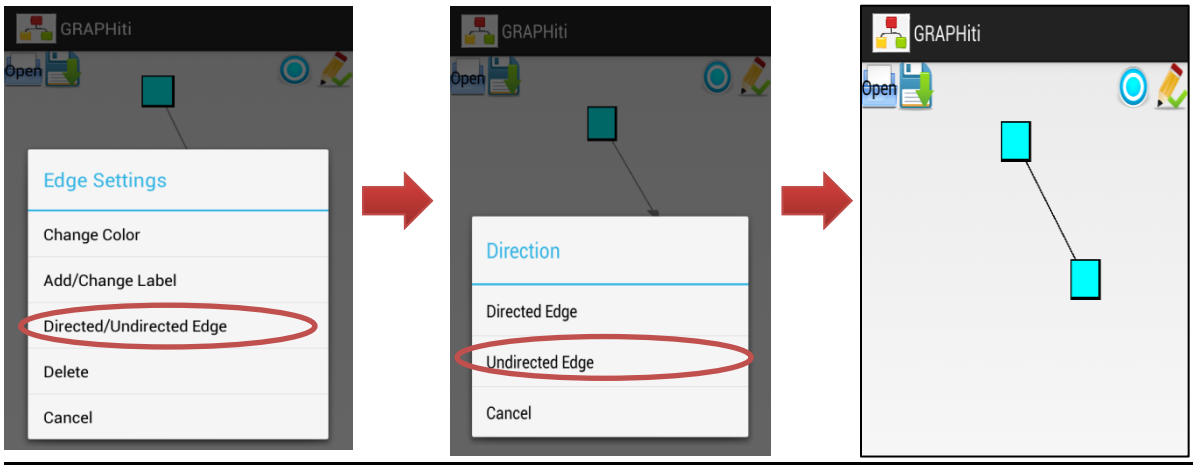
Εικόνα 21 Εμφάνιση διαλόγου ειδοποίησης κατά το long press σε μία ακμή.

Αλλαγή χρώματος ακμής

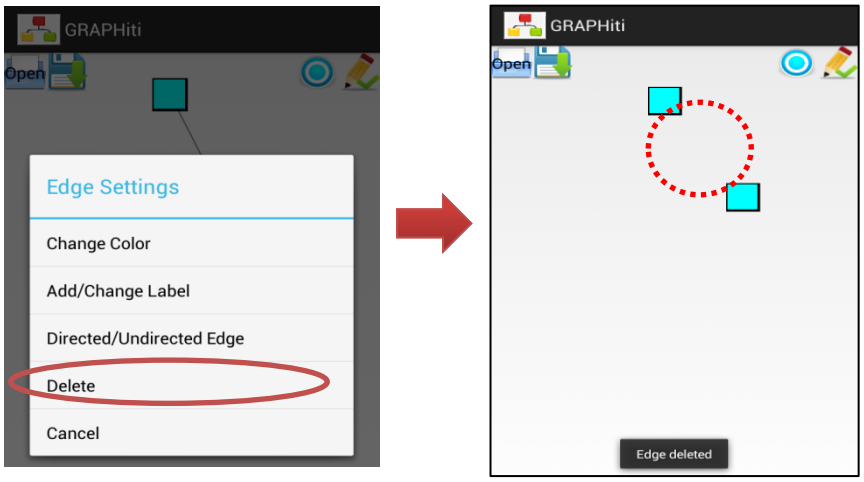


Προσθήκη ετικέτας κειμένου σε ακμή

Αλλαγή σε κατευθυνόμενη ή μη κατευθυνόμενη ακμή

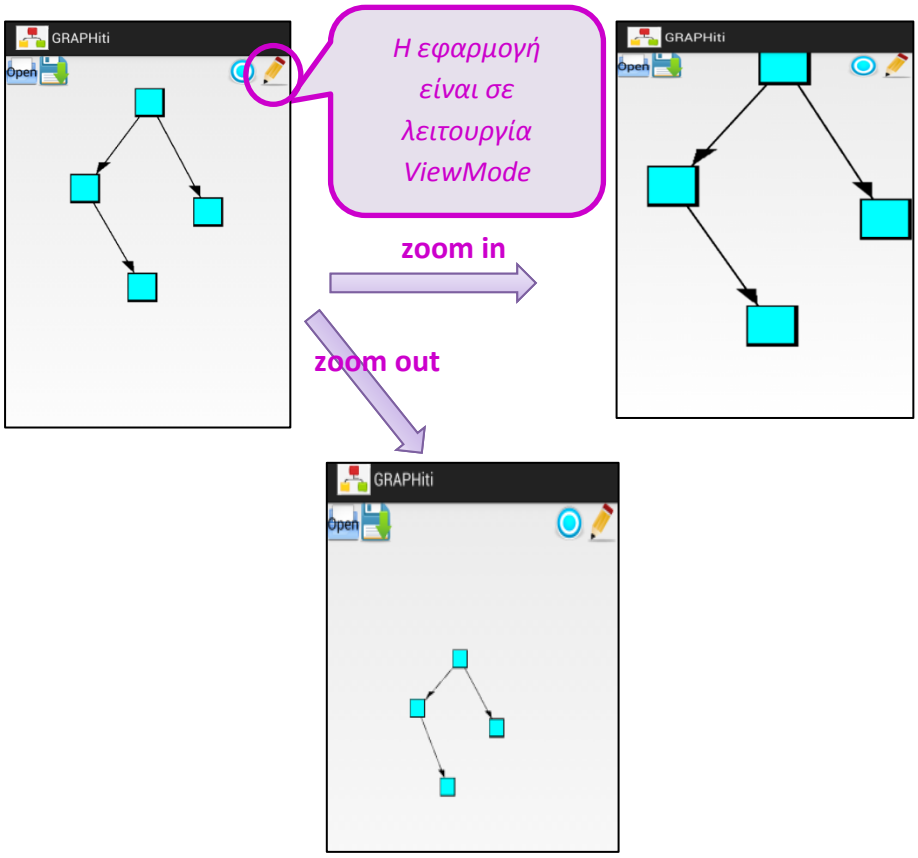


Διαγραφή ακμής



5.3. Zooming και Panning

Με τη μέθοδο «onScale(GestureDetector detector)» του *OnScaleGestureListener*, παρέχεται η λειτουργία «pinch zoom» όταν η εφαρμογή είναι σε λειτουργία «ViewMode», με την οποία επιτυγχάνεται η μεγέθυνση ή η σμίκρυνση του canvas στο οποίο απεικονίζεται το γράφημα



Επιπρόσθετα όταν η εφαρμογή είναι σε λειτουργία ViewMode, με το σύρσιμο του δακτύλου στην οθόνη επιτυγχάνεται η κύλιση του canvas, ενώ με διπλό πάτημα στην οθόνη γίνεται επαναφορά του ορατού μέρους της οθόνης στις αρχικές συντεταγμένες, δηλαδή έτσι ώστε η άνω αριστερή γωνία να αντιστοιχεί στο σημείο (0,0).

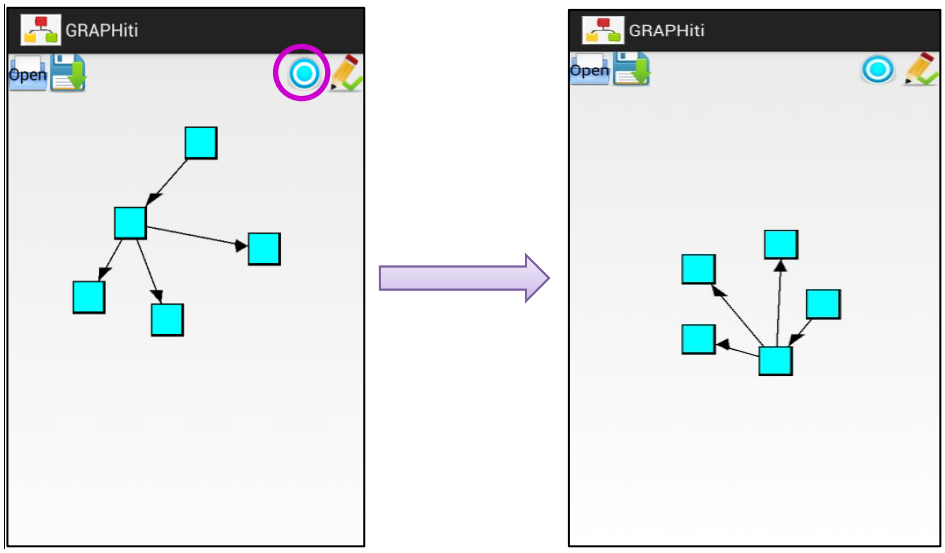
5.4. Αυτόματη διάταξη των κόμβων σε κύκλο

Με το πάτημα του κουμπιού  , οι κόμβοι του γραφήματος διατάσσονται αυτόματα σε κύκλο με ακτίνα ανάλογη του πλήθους τους. Αν οι κόμβοι του γραφήματος είναι λιγότεροι από έξι, τότε η ακτίνα του κύκλου είναι ίση με 100, αλλιώς είναι ίση με 210.

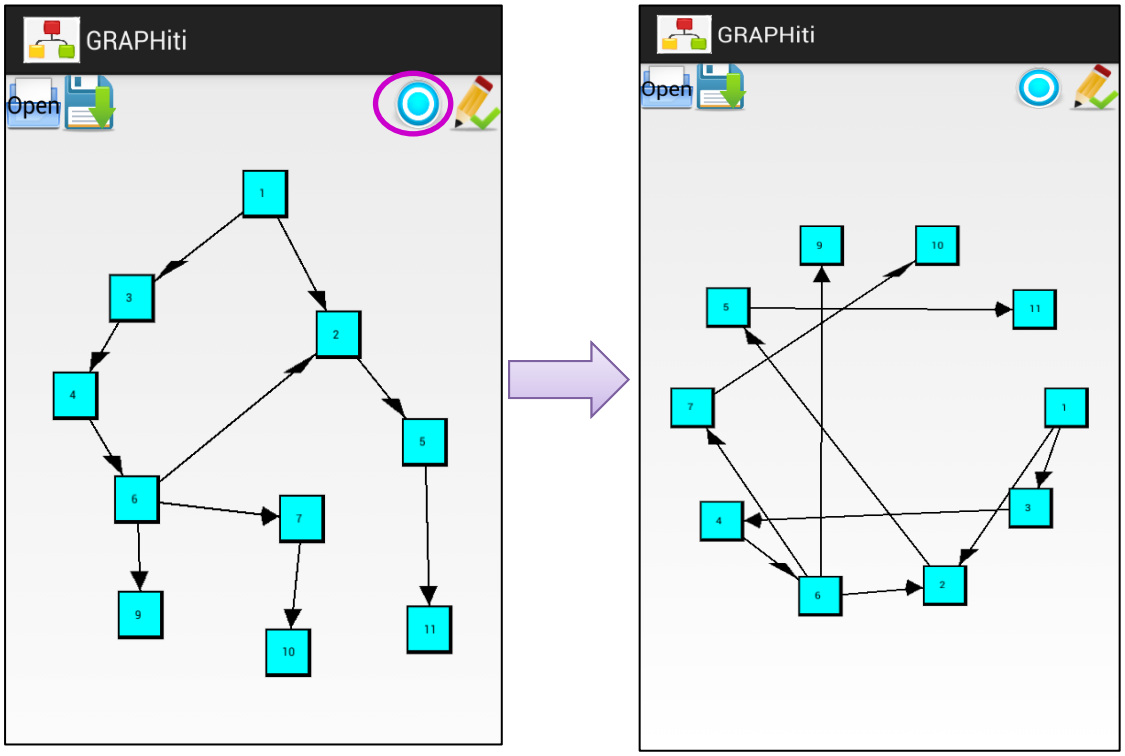
```
//graphView is of type 'VisualGraphView'
public void circularLayout(View view){
int noOfNodes=graphView.getEmbeddedGraph().nodeCount();
float centerX=270;
float centerY=425;
float radius;
if (noOfNodes<=5)
            radius=100;
 else
            radius=210;
float angle=((float)Math.PI*360/noOfNodes)/180;
float newAngle=0;
for (NodeIterator nodeIterator = graphView.getEmbeddedGraph().nodes();
nodeIterator.hasNextNode();)
        {
           Node currentNode = nodeIterator.nextNode();
           float newCenterX=centerX + (float)(radius*(Math.cos(newAngle)));
           float newCenterY=centerY + (float)(radius*(Math.sin(newAngle)));
           float
nodeWidth=graphView.getEmbeddedGraph().getNodeVisualizer(currentNode).getWidth();
           float
nodeHeight=graphView.getEmbeddedGraph().getNodeVisualizer(currentNode).getHeight();

graphView.getEmbeddedGraph().getNodeVisualizer(currentNode).setCenterX(newCenterX);

graphView.getEmbeddedGraph().getNodeVisualizer(currentNode).setCenterY(newCenterY);
           graphView.getEmbeddedGraph().getNodeVisualizer(currentNode).setX(newCenterX-
nodeWidth/2);
           graphView.getEmbeddedGraph().getNodeVisualizer(currentNode).setY(newCenterY-
nodeHeight/2);
           graphView.getEmbeddedGraph().getNodeVisualizer(currentNode).setBoundingBox();
           newAngle=newAngle+angle;
        }
 }
}
```

Εικόνα 22 Κώδικας για την αυτόματη διάταξη των κόμβων σε κύκλο.

Παρακάτω φαίνονται δύο παραδείγματα γραφημάτων όπου γίνεται αυτόματη διάταξη των κόμβων σε κύκλο.



Παράδειγμα 1



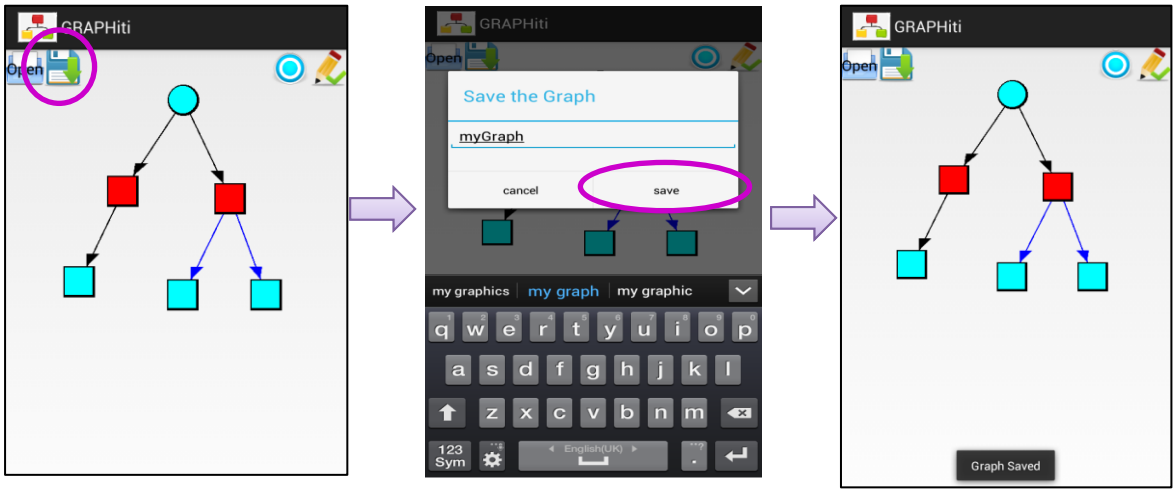
Παράδειγμα 2

5.5. Αποθήκευση και Ανάκτηση ενός γραφήματος

Η αποθήκευση και η ανάκτηση γραφημάτων από ένα αρχείο GraphML, επιτυγχάνεται με το πάτημα των αντίστοιχων κουμπιών που βρίσκονται στην άνω αριστερή γωνία της οθόνης.
Για την αποθήκευση, ένα κουτί μορφοποίησης κειμένου εμφανίζεται στην οθόνη ώστε να εισάγει ο χρήστης το όνομα του αρχείου που πρόκειται να αποθηκευθεί. Αντίστοιχα, για το άνοιγμα ενός αρχείου το οποίο περιέχει γράφημα σε GraphML, εμφανίζεται στην οθόνη ένα παράθυρο με τους υπάρχοντες φακέλους της συσκευής ώστε να μπορεί ο χρήστης να πλοηγηθεί και να επιλέξει το αρχείο το οποίο θέλει να φορτώσει.
*Σημείωση: Για τη διαδικασία φόρτωσης ενός γραφήματος χρησιμοποιήθηκε η βιβλιοθήκη **iPaulPro/ aFileChooser,** με τη βοήθεια της οποίας επιτυγχάνεται η πλοήγηση στα αρχεία της συσκευής ώστε να μπορεί ο χρήστης να εντοπίσει το αρχείο το οποίο θέλει να φορτώσει.*

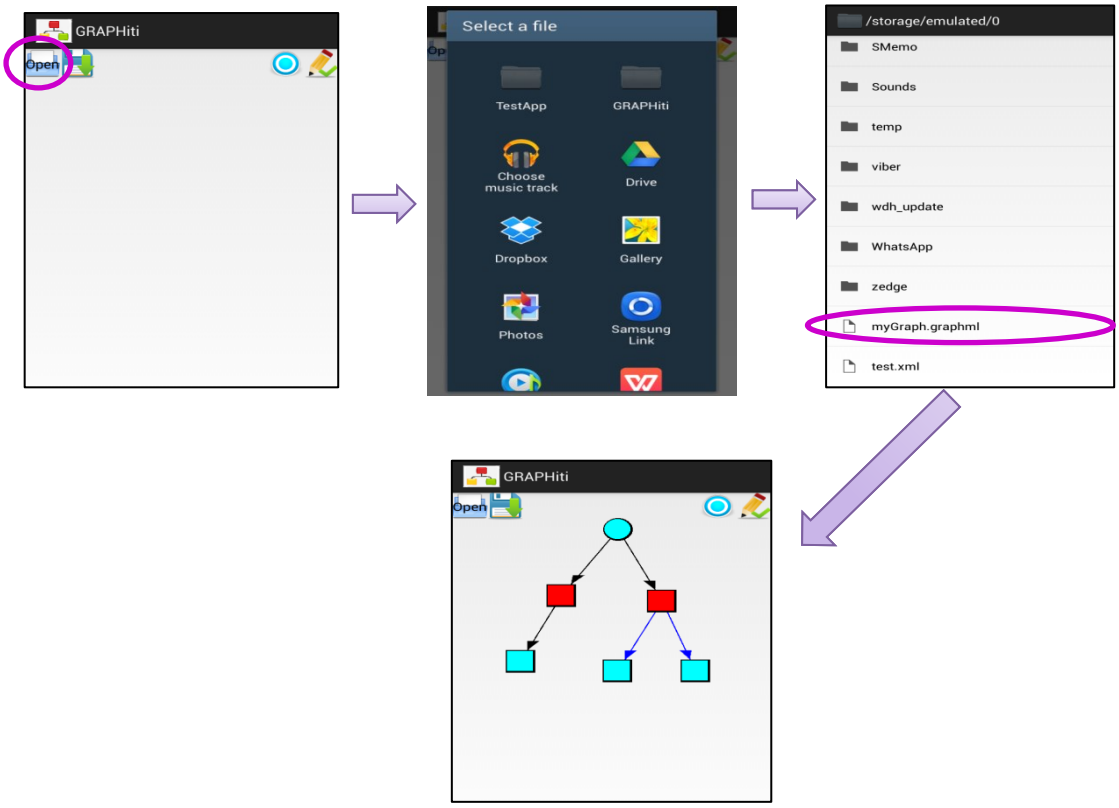
```
final EditText  name=new EditText(view.getContext());
String fileName=name.getText().toString();
File file=new File("/storage/emulated/0" + File.separator + fileName + ".graphml");
OutputStream out=null;
try {
        out = new BufferedOutputStream(new FileOutputStream(file));
        SaveToGraphML.saveTographML(graphView.getEmbeddedGraph(),out);
        Toast.makeText(graphView.getContext(), "Graph Saved",
Toast.LENGTH_SHORT).show();
                }
```

Εικόνα 23 Απόσπασμα κώδικα της εφαρμογής για την αποθήκευση γραφήματος.



```
File file=new File(path);
 InputStream in=null;
try{
        in = new BufferedInputStream(new FileInputStream(file));
        LoadFromGraphML.loadFromGraphML(graphView.getEmbeddedGraph(),in);
}
```

Εικόνα 24 Απόσπασμα κώδικα της εφαρμογής για την φόρτωση ενός γραφήματος.



Κεφάλαιο 6: ΣΥΝΟΨΗ

Η δημιουργία εφαρμογών για απεικόνιση γραφημάτων στα κινητά τηλέφωνα είναι ακόμη σε σχετικά αρχικά στάδια και έχει αρκετό περιθώριο εξέλιξης.
Στην παρούσα διπλωματική εργασία έγινε προσπάθεια για τη δημιουργία εφαρμογής Android, η οποία αποσκοπεί στο σχεδιασμό γραφημάτων διατηρώντας πληροφορίες για τη δομή του γραφήματος ως αυστηρή μαθηματική έννοια, και η οποία χρησιμοποιεί τις ήδη υπάρχουσες τεχνικές στο Android, όπως είναι για παράδειγμα το «pinch zooming», με τέτοιο τρόπο ώστε να ελαχιστοποιούνται τα προβλήματα που αντιμετωπίζει κάποιος ο οποίος θέλει να αναπαραστήσει ένα γράφημα σε κινητό τηλέφωνο. Ένα κύριο πρόβλημα είναι για παράδειγμα το μικρό μέγεθος της οθόνης, που περιορίζει τη λειτουργικότητα τέτοιων εφαρμογών. Επίσης, έγινε προσπάθεια για την ανάπτυξη διεπαφής χρήστη υψηλής ποιότητας.

Η εφαρμογή GRAPHiti καθώς και η βιβλιοθήκη που αναπτύχθηκε επιδέχονται βελτιώσεις. Ειδικότερα, στη βιβλιοθήκη μπορούν να προστεθούν υλοποιήσεις των realizers των κόμβων και των ακμών, δηλαδή για παράδειγμα να υποστηρίζεται κόμβος σε μορφή εικόνας, και καμπύλες και πολυγωνικές ακμές. Ενώ σε επίπεδο εφαρμογής μπορούν να προστεθούν περαιτέρω λειτουργίες για υποστήριξη περισσότερων δυνατοτήτων στην αλληλεπίδραση του χρήστη με το γράφημα.

Γενικά, μέσα από την εκπόνηση της διπλωματικής εργασίας απέκτησα σημαντικές και πολύτιμες γνώσεις. Ήταν μια εμπειρία μέσα από την οποία έμαθα να αναπτύσσω τεχνικές για να επιλύω τα προβλήματα και να δείχνω αποφασιστικότητα για το ξεπέρασμα των εμποδίων και των δυσκολιών που προκύπτουν. Είχα επίσης τη δυνατότητα να αξιοποιήσω τις ήδη υπάρχουσες γνώσεις μου στον προγραμματισμό αλλά και να τις αναπτύξω περαιτέρω. Επιπλέον, ήταν μία ευκαιρία για μένα να αξιοποιήσω τις γνώσεις μου σε δημιουργικές εφαρμογές, αντιπροσωπεύοντας έτσι και την κυριολεκτική έννοια που αντιπροσωπεύει η σχολή μου, την ενασχόληση με εφαρμογές των Μαθηματικών.

Βιβλιογραφία

[1] U. Doğrusöz και G. Sander, «Graph Visualization».

[2] G. Da Lozzo, G. Di Batista και F. Ingrassia, «Drawing Graphs on a Smartphone,» *Journal of Graphs Algorithms and Applications,* p. 18, 2012.

[3] G. Di Battista, P. Eades, R. Tamassia και I. . G. Tollis, «Algorithms for Drawing Graphs: an Annotated Bibliography,» 1994.

[4] M. Kaufmann και D. Wagner, Drawing Graphs Methods and Models, Springer, 2001.

[5] M. Gargenta, Learning Android, O'Reilly, 2011.

[6] GraphML Team, «graphdrawing.org,» [Ηλεκτρονικό]. Available: http://graphml.graphdrawing.org/.

[7] Android Developers, «Canvas,» [Ηλεκτρονικό]. Available: http://developer.android.com/reference/android/graphics/Canvas.html.

[8] Android Developers, «View,» [Ηλεκτρονικό]. Available: http://developer.android.com/reference/android/view/View.html.

[9] Android Developers, «Dragging and Scaling,» [Ηλεκτρονικό]. Available: http://developer.android.com/training/gestures/scale.html.

[10] yWorks, «yFiles for Java,» [Ηλεκτρονικό]. Available: http://www.yworks.com/en/products_yfiles_about.html.

[11] yWorks , «GraphML,» [Ηλεκτρονικό]. Available: www.yworks.com/en/products_yfiles_ep_graphml.htm.

[12] Rough Book, «Implementing pinch-zoom and pan/drag in an Android view on the canvas,» 11 Dec 2011. [Ηλεκτρονικό]. Available: http://vivin.net/2011/12/04/implementing-pinch-zoom-and-pandrag-in-an-android-view-on-the-canvas/.

[13] R. Tamassia και I. Cruz, «Graph Drawing Tutorial».

[14] B. Lahti, «Multitouch Panning and Zooming Examples for Android,» 7 Jan 2013. [Ηλεκτρονικό]. Available: http://blahti.wordpress.com/2013/01/07/pan-zoom-examples-for-android/.

[15] Android Developers, «Getting Started,» [Ηλεκτρονικό]. Available: http://developer.android.com/training/index.html.

[16] iPaulPro, «aFileChooser,» 2014.

[17] Android Developers, «Get the Android SDK».

Παράρτημα: Κώδικας της Εφαρμογής GRAPHiti

```
package com.example.graphiti;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import basic.NodeIterator;
import com.ipaulpro.afilechooser.utils.FileUtils;
import basic.Node;
import visual.DefaultVisualEditMode;
import visual.DefaultVisualViewMode;
import visual.LoadFromGraphML;
import visual.SaveToGraphML;
import visual.VisualGraphView;
import visual.VisualModeListener;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.widget.CompoundButton;
import android.widget.EditText;
import android.widget.Toast;
import android.widget.ToggleButton;

public class MainActivity extends Activity {
        VisualGraphView graphView;
        VisualModeListener editMode,viewMode;
        ToggleButton changeModesButton;


        /**
     * Called when the activity is first created.
     * @param savedInstanceState, the bundle
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);
         setContentView(R.layout.activity_main);
         graphView = (VisualGraphView)findViewById(R.id.graph_view_editMode);

         editMode=new DefaultVisualEditMode(graphView);

         graphView.addVisualMode(editMode);
         graphView.setViewModeOn(false);

        viewMode=new DefaultVisualViewMode(graphView);
        changeModesButton=(ToggleButton)findViewById(R.id.editModeButton);
        changeModesButton.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
```

```
            public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
                    if (isChecked) {
                         graphView.addVisualMode(editMode);
                         graphView.removeVisualMode(viewMode);
                         graphView.setViewModeOn(false);
                    }
                    else {
                         graphView.removeVisualMode(editMode);
                         graphView.addVisualMode(viewMode);
                         graphView.setViewModeOn(true);
                    }
                       }
        });
    }


        /**
          * Saves the graph into a GraphML file
          */
    public void saveDiagram(View view){
        final EditText  name=new EditText(view.getContext());
        new AlertDialog.Builder(view.getContext())
        .setTitle("Save the Graph")
        .setView(name)
        .setPositiveButton("save", new DialogInterface.OnClickListener() {
          @Override
          public void onClick(DialogInterface dialog, int id) {
                 String fileName=name.getText().toString();
                 File file=new File("/storage/emulated/0" + File.separator +
fileName + ".graphml");
                 OutputStream out=null;
                try {
                   out = new BufferedOutputStream(new FileOutputStream(file));
                   SaveToGraphML.saveTographML(graphView.getEmbeddedGraph(),out);
                   Toast.makeText(graphView.getContext(), "Graph Saved",
Toast.LENGTH_SHORT).show();
                 }
//              finally {
//              }

                 catch (IllegalArgumentException e) {
                             // TODO Auto-generated catch block
                             e.printStackTrace();
                      }
                 catch (IllegalStateException e) {
                             // TODO Auto-generated catch block
                             e.printStackTrace();
                      }
                 catch (IOException e) {
                             // TODO Auto-generated catch block
                             e.printStackTrace();
                      }

        }})
        .setNegativeButton("cancel", new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int id) {
                 Toast.makeText(graphView.getContext(), "Cancelled",
Toast.LENGTH_SHORT).show();
          }
        })
        .show();
}

    private static final int REQUEST_CHOOSER= 1234;
```

```
        protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
                switch (requestCode) {
                    case REQUEST_CHOOSER:
                        if (resultCode == RESULT_OK) {

                            final Uri uri = data.getData();

                            // Get the File path from the Uri
                            String path = FileUtils.getPath(this, uri);
                            File file=new File(path);
                            InputStream in=null;
                   try{
                     in = new BufferedInputStream(new FileInputStream(file));

LoadFromGraphML.loadFromGraphML(graphView.getEmbeddedGraph(),in);
                                    }
                                catch (IOException e){
                                            new AlertDialog.Builder(this)
                                    .setIcon(android.R.drawable.ic_dialog_alert)
                                    .setTitle("Open from")
                                    .setMessage("Error occured while opening");
                                 }

                            }
                            break;
                    }
                }
/**
* Loads the graph
*/
public void openDiagram(View view){
// Create the ACTION_GET_CONTENT Intent
        graphView.getEmbeddedGraph().clear();
        graphView.setTranslateX(0);
        graphView.setTranslateY(0);
        Intent getContentIntent = FileUtils.createGetContentIntent();

        Intent intent = Intent.createChooser(getContentIntent, "Select a file");
        startActivityForResult(intent, REQUEST_CHOOSER);
}

/**
* Called when a key down event has occurred.
* @param event, The KeyEvent object that defines the button action.
* @param keyCode, A key code that represents the button pressed, from KeyEvent.
* @return If you handled the event, return true. If you want to allow the event to
be handled by the next receiver, return false.
*/
public boolean onKeyDown(int keyCode, KeyEvent event)
{
//           Handle the back button
        if(keyCode == KeyEvent.KEYCODE_BACK)
        {
//                  Ask users if they want to quit
        new AlertDialog.Builder(this)
        .setIcon(android.R.drawable.ic_dialog_alert)
        .setTitle("Exit")
        .setMessage("Are you sure you want to Exit ?")
        .setPositiveButton("Yes", new DialogInterface.OnClickListener()
        {
        public void onClick(DialogInterface dialog, int which)
        {
        Intent intent = new Intent(Intent.ACTION_MAIN);
        intent.addCategory(Intent.CATEGORY_HOME);
```

```
        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        startActivity(intent);
        finish();
        }
        })
        .setNegativeButton("No", null)
        .show();
        return true;
        }
        else
        {
                return super.onKeyDown(keyCode, event);
        }
        }

/**
* Arrange the nodes in a circle
* @param view, the selected view
*/
public void circularLayout(View view){
        int noOfNodes=graphView.getEmbeddedGraph().nodeCount();
        float centerX=270;
        float centerY=425;
        float radius;
        if (noOfNodes<=5){
                radius=100;
        }
        else
                radius=210;
        float angle=((float)Math.PI*360/noOfNodes)/180;
        float newAngle=0;
        for (NodeIterator nodeIterator = graphView.getEmbeddedGraph().nodes();
        nodeIterator.hasNextNode();)
        {
        Node currentNode = nodeIterator.nextNode();
        float newCenterX=centerX + (float)(radius*(Math.cos(newAngle)));
        float newCenterY=centerY + (float)(radius*(Math.sin(newAngle)));
        float
        nodeWidth=graphView.getEmbeddedGraph().getNodeVisualizer(currentNode).getWid
        th();
        float
        nodeHeight=graphView.getEmbeddedGraph().getNodeVisualizer(currentNode).getHe
        ight();
        graphView.getEmbeddedGraph().getNodeVisualizer(currentNode).setCenterX(newCe
        nterX);
        graphView.getEmbeddedGraph().getNodeVisualizer(currentNode).setCenterY(newCe
        nterY);
        graphView.getEmbeddedGraph().getNodeVisualizer(currentNode).setX(newCenterX-
        nodeWidth/2);
        graphView.getEmbeddedGraph().getNodeVisualizer(currentNode).setY(newCenterY-
        nodeHeight/2);
        graphView.getEmbeddedGraph().getNodeVisualizer(currentNode).setBoundingBox()
        ;
        newAngle=newAngle+angle;
}
}

}
```