ΑΤΟΜΙΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση Εργαλείου για Αλγοριθμική Σύνθεση Μουσικής Μπαρόκ με τη χρήση Γενετικών Αλγορίθμων

Τσόκκου Παναγιώτα

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2011

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΛΓΟΡΙΘΜΙΚΗ ΣΥΝΘΕΣΗ ΜΟΥΣΙΚΗΣ: ΓΕΝΕΤΙΚΟΙ ΑΛΓΟΡΙΘΜΟΙ

Τσόκκου Παναγιώτα

Επιβλέπουσα Καθηγητής

Άννα Φιλίππου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2011

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον ουσιαστικό επιβλέποντα της Διπλωματικής μου, την κ. Άννα Φιλίππου για την αμέριστη βοήθεια της και για το χρήσιμο συμβουλευτικό της έργο.
Επίσης θα ήθελα να ευχαριστήσω θερμά την οικογένεια μου για τη συμπαράσταση, την κατανόηση και την υπομονή τους σε όλη τη διάρκεια αυτής της εργασίας αλλά και των σπουδών μου.

Περίληψη

Η συγκεκριμένη διπλωματική εργασία ασχολείται με το θέμα της αλγοριθμικής σύνθεσης και είχε ως κύριο σκοπό την ανάπτυξη προγράμματος στο οποίο θα δημιουργούνται μελωδίες υιοθετώντας χαρακτηριστικά στοιχεία της εποχής Μπαρόκ χρησιμοποιώντας γενετικό αλγόριθμο. Στόχος της μελέτης που έγινε αφορά την κατανόηση του τρόπου ανάπτυξης των γενετικών αλγορίθμων.

Παρουσιάζουμε ένα γενετικό αλγόριθμο όπου βασίζεται στην εξής ιδέα: μέσω ενός αρχικού τυχαίου πληθυσμού μελωδιών εκτελούνται συγκεκριμένες γενετικές λειτουργίες, γίνονται αξιολογήσεις σε κάθε μελωδία, αναπαράγεται ο πληθυσμός, έτσι ώστε στο τέλος ο αλγόριθμος να καταλήξει στην επιλογή της μελωδίας με το καλύτερο αποτέλεσμα για να παρουσιαστεί και για να εκτελεστεί.

Στην ανάπτυξη του προγράμματος χρησιμοποιήθηκε η γλώσσα προγραμματισμού Java μέσω της οποίας υπήρξε η δυνατότητα να χρησιμοποιήσουμε την βιβλιοθήκη της Jmusic. Η συγκεκριμένη βιβλιοθήκη μας δίνει την δυνατότητα να δημιουργούμε μελωδίες, να τις εκτελούμε και να τις αποθηκεύουμε. Για την υλοποίηση της διεπαφής χρησιμοποιήθηκε το εργαλείο NetBeans.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Εισαγωγή

Μουσική είναι η τέχνη που εκφράζει συναισθήματα λύπης, χαράς, έκπληξης και αγάπης μέσω μιας ακολουθίας από ήχους. Κύριοι στόχοι της είναι η δημιουργία και η εκτέλεση μελωδιών. Αρκετοί άνθρωποι θεωρούν ότι η έννοια της μουσικής έχει την ίδια έννοια με τη μελωδία λόγω του γεγονότος ότι μια μελωδία αποτελείται ένα πεπερασμένο αριθμό νότων και όταν εκτελέσουμε την μελωδία με κάποιο μουσικό όργανο ακούγεται συνήθως ομοιόμορφος ήχος. Η διαδικασία της δημιουργίας μελωδιών ονομάζεται σύνθεση, όπου αυτή η διαδικασία απαιτεί από το συνθέτη τόσο φαντασία όσο και δημιουργικότητα. Συνήθως μια μελωδία δημιουργείται από τον ίδιο τον άνθρωπο. Παρ' όλα αυτά δεν είναι λίγα τα συστήματα τα οποία έχουν αναπτυχθεί με κύριο στόχο τη δημιουργία μελωδίων χωρίς καμία βοήθεια από κάποιο συνθέτη. Η διαδικασία στην οποία δημιουργούνται συνθέσεις χρησιμοποιώντας μόνο τους ηλεκτρονικούς υπολογιστές ονομάζεται αλγοριθμική σύνθεση. Οι ηλεκτρονικοί υπολογιστές δίνουν τη δυνατότητα να δημιουργήσουν μελωδίες χρησιμοποιώντας διάφορες τεχνικές όπως κανόνες, συστήματα με περιορισμούς, πολύπλοκα μαθηματικά μοντέλα και διάφορες άλλες μεθόδους.

Στη συγκεκριμένη εργασία θα αναπτυχθεί σύστημα το οποίο δημιουργεί μελωδίες, οι οποίες περιέχουν χαρακτηριστικά στοιχεία της εποχής Μπαρόκ, χρησιμοποιώντας γενετικό

αλγόριθμο. Το πρόγραμμα στο τέλος θα έχει τη δυνατότητα να εκτελεί τη μελωδία που δημιούργησε.

1.2 Ιστορική αναδρομή

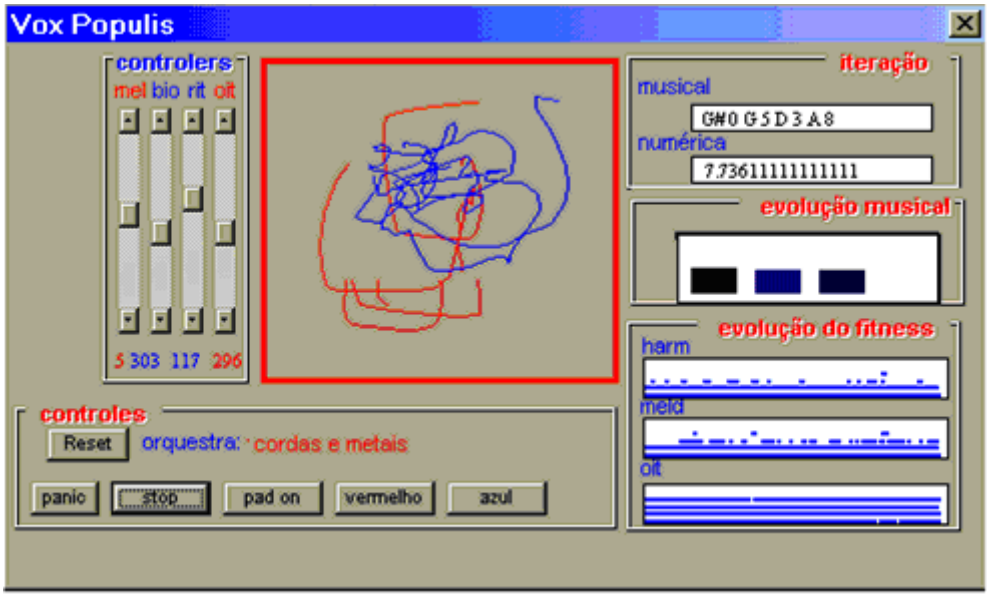
Τα τελευταία χρόνια η αλγοριθμική σύνθεση έχει αναπτυχθεί αρκετά προτείνοντας συνεχώς καινούργιες τεχνικές [1]. Όμως οι πρώτες προσεγγίσεις έχουν προωθηθεί εδώ και χρόνια. Αρχικά, τον 11^{ο} αιώνα ο Guido d' Arrezo είχε δημιουργήσει μια τεχνική μέσω της οποίας σε ένα δοσμένο κείμενο, όταν συναντούσε φωνήεν στο κείμενο σημείωνε και μια νότα, δημιουργώντας με αυτό τον τρόπο μια μελωδία. Ένα αρκετά γνωστό σύστημα αλγοριθμικής σύνθεσης είναι το Mozart Dice Game το οποίο δημοσιεύθηκε για πρώτη φορά το 1793, στο οποίο ο Μότσαρτ μέσω ενός συνόλου κανόνων είχε συνθέσει μινουέτα. Με βάση αυτό το σύστημα ο Μότσαρτ είχε εισάγει την έννοια της επιλογής και της τύχης αρκετό διάστημα πριν χρησιμοποιηθεί στους ηλεκτρονικούς υπολογιστές. Μηχανή η οποία είχε τη δυνατότητα να δημιουργεί αυτοσχέδιες μελωδίες ήταν η "arca Musirithmica" η οποία δημιουργήθηκε από τον Athanasius Kirchner το 1660. Ακολούθησε το 1821 από τον Dietrich το σύστημα 'Winkel Componium', ενώ το 1950 οι Harry Olsen και ο Hebert Belar δημιούργησαν μια ηλεκτρομηχανική μηχανή στην οποία δημιουργούσαν μελωδίες στηριζόμενες στην πιθανότητα.

Στις μέρες μας, η αλγοριθμική σύνθεση χρησιμοποιείται όλο και πιο συχνά από διάφορους συνθέτες για να επιτύχουν τον σκοπό της σύνθεσης. Ο στόχος κατά τη διαδικασία της αλγοριθμικής σύνθεσης είναι ο αλγόριθμος να συμπεριφέρεται όπως θα συμπεριφερόταν ο συνθέτης κατά τη διάρκεια της δημιουργίας μιας μελωδίας. Αρκετά λογισμικά αναπτύσσονται συνεχώς για να εξυπηρετήσουν τους συνθέτες.

Λογισμικά όπως το Vox Populi που παράγει μελωδίες οι οποίες μπορούν στη συνέχεια να τροποποιηθούν από το συνθέτη αλλάζοντας μερικές νότες είτε ρυθμό δίνοντας του έτσι την ευκαιρία να δώσει μερικά στοιχεία από το στυλ του [2, 3]. Το συγκεκριμένο λογισμικό χρησιμοποιεί τους γενετικούς αλγορίθμους για να εκτελέσει τους σκοπούς του. Είναι λογισμικό για παραγωγή μουσικών κομματιών που χρησιμοποιείται σε Windows. Δημιουργεί ένα μοτίβο από συγχορδίες, όπου η κάθε συγχορδία αποτελείται από 4 νότες και αναπαριστάται ως μια 7-bit συμβολοσειρά. Μετά η κάθε συγχορδία αντιπροσωπεύεται από μια 28-bit συμβολοσειρά και γίνονται οι διαδικασίες της διασταύρωσης και μεταλλαγής για να δημιουργηθεί νέος πληθυσμός. Αυτή η διαδικασία επαναλαμβάνεται μέχρι να έχουμε μια ικανοποιητική λύση. Το μοτίβο αξιολογείται με βάση τη μελωδία, την αρμονία και τις οκτάβες

των νότων. Χρησιμοποιείται για πραγματικό χρόνο παρουσίαση της μελωδίας και ο χρήστης μπορεί να αλλάξει τη μουσική χρησιμοποιώντας το ποντίκι, τροποποιώντας τον τόνο και τα όρια των φωνών.

Ο χρήστης μπορεί να παρέμβει στη λειτουργία ικανότητας μέσω τεσσάρων ελέγχων. Ο πρώτος έλεγχος γίνεται συγκρίνοντας τις νότες της μελωδίας με ένα id το οποίο προσφέρει τον έλεγχο της διεπαφής, μπορεί να τροποποιηθεί από το χρήστη και χρησιμοποιείται για να αξιολογήσει τη μελωδική ικανότητα. Εκτελώντας αυτόν τον έλεγχο ο χρήστης αλλάζει τον τόνο του μουσικού κομματιού. Ο δεύτερος έλεγχος παρεμβαίνει στη διάρκεια του γενετικού κύκλου, ο χρήστης τροποποιεί το ρυθμό, καθιστώντας το γρηγορότερο ή πιο αργό. Ο τρίτος έλεγχος αφορά τις νότες που αποτελούν τη συγχορδία αν είναι εντός των επιτρεπτών ορίων, δηλαδή αν ανήκουν στις επιλεγμένες οκτάβες. Ο τέταρτος έλεγχος αφορά το χρονικό τμήμα που θα παίζουν τους ήχους κάθε ορχήστρα.

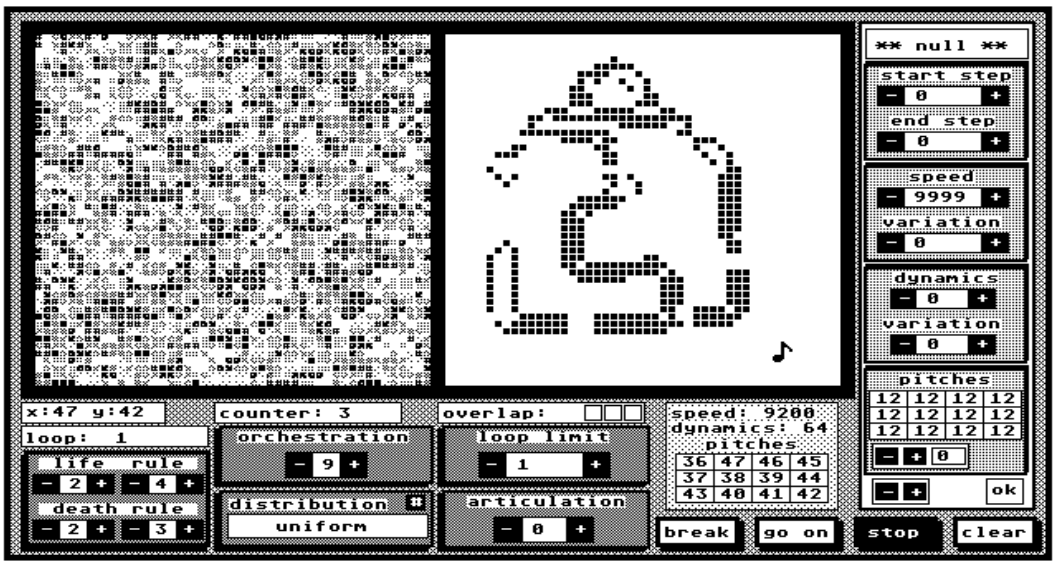


Εικόνα 1: Παράδειγμα εκτέλεσης του Vox Populi

Ένα άλλο λογισμικό το οποίο χρησιμοποιείται αρκετά συχνά από τους συνθέτες είναι το CAMUS το οποίο χρησιμοποιεί τα κυψελοειδή αυτόματα για την παραγωγή μουσικού κομματιού σε μορφή MIDI αρχείου [1, 4]. Χρησιμοποιεί δύο ειδών αυτόματα. Το Game of Life το οποίο αποτελείται από ένα πίνακα από κελιά που το καθένα μπορεί να είναι ζωντανό ή νεκρό. Το ζωντανό παίρνει την τιμή 1 και το νεκρό την τιμή 0. Σε κάθε βήμα το κελί το οποίο έχει τους 3 με 4 γείτονες του ζωντανούς τότε παραμένει ζωντανό και το νεκρό κελί γίνεται ζωντανό εάν έχει τους 3 γείτονες του ζωντανούς. Στο τέλος τα ζωντανά κελιά απεικονίζονται

μαύρα ενώ τα νεκρά είναι λευκά. Το δεύτερο αυτόματο που χρησιμοποιεί το λογισμικό CAMUS είναι το Demon Cyclic Space. Είναι και αυτός ένας πίνακας κελιών στον οποίο κάθε φορά ένα κελί εξουσιάζει άλλα κελιά τα οποία είναι στην προηγούμενη κατάσταση μετατρέποντας τα στο ίδιο χρώμα με αυτό.

Στο CAMUS και τα δύο αυτά αυτόματα εκτελούνται παράλληλα, με το Game of Life να παράγει τις νότες, ενώ το Demon Cyclic Space είναι υπεύθυνο για την ενορχήστρωση της σύνθεσης όπου κάθε όργανο που χρησιμοποιείται στην εκτέλεση του μουσικού κομματιού αντιπροσωπεύεται από ένα χρώμα. Αρχικά, ξεκινούν και τα δύο αυτόματα να τρέχουν και κάθε φορά αναλύονται οι συντεταγμένες κάθε ζωντανού κελιού για να καθοριστούν οι νότες που θα αποτελέσουν τις συγχορδίες του μουσικού κομματιού. Μετά εντοπίζεται από το αντίστοιχο κελί στο Demon Cyclic Space το όργανο το οποίο θα εκτελέσει τη συγκεκριμένη συγχορδία. Αυτό επαναλαμβάνεται για όλα τα κελιά υπολογίζοντας τον τόνο και τη διάρκεια της κάθε συγχορδίας τριών νότων προσωρινά αφού στη συνέχεια ο χρήστη μπορεί να δώσει μια δική του εξίσωση για να υπολογιστούν.



Εικόνα 2: Παράδειγμα εκτέλεσης του λογισμικού CAMUS

Ακόμη ένα λογισμικό το οποίο συνηθίζεται να χρησιμοποιείται είναι το GenJam το οποίο είναι υπεύθυνο για σύνθεση κυρίως jazz μουσικών κομματιών και το οποίο χρησιμοποιεί γενετικούς αλγορίθμους [3, 5]. Αρχικά διαβάζει ένα αρχείο το οποίο καθορίζει το ρυθμό, τον τόνο, την οκτάβα με την οποία θα ξεκινήσει η μελωδία και ένα μοτίβο από χορδές. Ακόμη του δίνονται ως είσοδος τα μουσικά όργανα που θα χρησιμοποιήσει σε κάθε μέρος και πόσο δυνατά

επιθυμεί να ακουστούν. Το GenJam μπορεί να ακούσει μουσική και να το χρησιμοποιήσει για τη δημιουργία μουσικού κομματιού.
Όπως είναι προφανές, οι συνθέτες δεν χρησιμοποιούν πλέον τόσο συχνά την παραδοσιακή μέθοδο για τη σύνθεση μελωδιών, αλλά η αλγοριθμική σύνθεση έχει πρωταγωνιστικό ρόλο στη ζωή τους. Επομένως, χρειάζεται περισσότερη ανάπτυξη όλων και περισσότερων λογισμικών τα οποία εξυπηρετούν τις ανάγκες των ανθρώπων στη σύνθεση.

1.3 Σύγχρονοι συνθέτες της αλγοριθμικής σύνθεσης

Αρκετοί συνθέτες και ερευνητές τις τελευταίες δεκαετίες χρησιμοποιούν τους ηλεκτρονικούς υπολογιστές είτε για να αναπτύξουν λογισμικά για δημιουργία συνθέσεων, είτε για να χρησιμοποιήσουν τα συστήματα για να εξυπηρετήσουν τις ανάγκες τους για σύνθεση. Μερικοί σημαντικοί πρωτοπόροι της αλγοριθμικής σύνθεσης οι οποίοι συνθέτανε με τη χρήση των ηλεκτρονικών υπολογιστών ήταν οι Ιάννης Ξενάκης, ο Barry Truax, ο Lejaren Hiller.

Ο Ιάννης Ξενάκης ήταν συνθέτης ο οποίος στην αρχή της καριέρας του σύνθετε μελωδίες όπως συνηθίζουν οι περισσότεροι συνθέτες, δηλαδή ανάπτυσσε τις μουσικές του ιδέες στο χέρι [1, 4, 6]. Συγκεκριμένα το 1955 είχε παρουσιάσει μια σύνθεση την οποία δημιούργησε χωρίς τη χρήση ηλεκτρονικών υπολογιστών τη λεγόμενη "Metastasis", η οποία ήταν γραμμένη για ορχήστρα και είχε μεγάλη απήχηση στον κόσμο. Ταυτόχρονα είχε ξεκινήσει να εισάγει την ιδέα της στοχαστικής μουσικής μέσω της οποίας θεωρίες των μαθηματικών που έχουν σχέση με τις πιθανότητες μεταφέρονται στη μουσική. Το 1961 είχε ξεκινήσει να επεκτείνει τις ιδέες του στη μουσική χρησιμοποιώντας τους ηλεκτρονικούς υπολογιστές αυτή την φορά για να δημιουργεί συνθέσεις. Οι πρώτες του συνθέσεις που είχε δημιουργήσει μέσω των ηλεκτρονικών υπολογιστών ήταν οι "ST/48-1,240162", "Amorsima-Morsima", και η "Atres". Το 1962 είχε χρησιμοποιήσει το πρόγραμμα SMP (Stochastic Music Program) το οποίο δημιουργούσε μελωδίες ως μια πρόταση αποτελούμενη από τμήματα, όπου το κάθε τμήμα είχε διαφορετική διάρκεια και πυκνότητα από νότες.

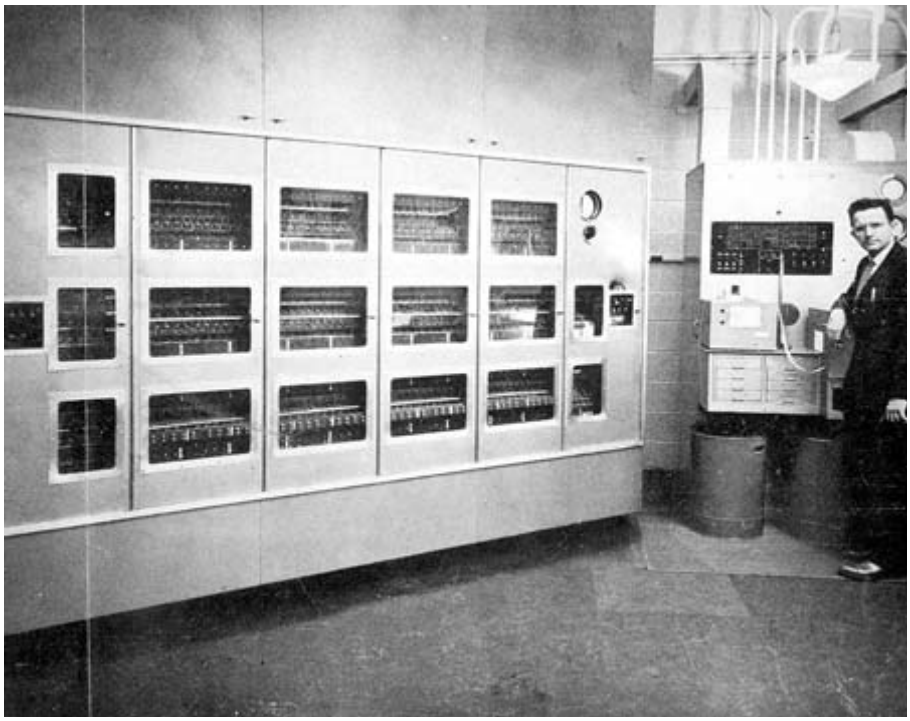
Εικόνα 3: Ο Ιάννης Ξενάκης

Ο Gottfried Michael Konig το 1964 είχε αναπτύξει το λογισμικό σύνθεσης Project 1 το οποίο βασιζόταν στη θεωρία της σειριακής μουσικής, στην οποία έχοντας μια σειρά από νότες δημιουργείται σύνθεση η οποία περιέχει τη συγκεκριμένη σειρά και την αντίστροφη της, δηλαδή ξεκινώντας από το τέλος προς την αρχή, με τη διαφορά ότι προσπάθησε να αντικαταστήσει την ιδέα των σειρών με μουσικές αξίες [1]. Το σύστημα αυτό επέστρεφε ως έξοδο μια λίστα με νότες όπου λίστα βασιζόταν σε διάφορες παραμέτρους όπως είναι ο ρυθμός, η αρμονία και η δυναμικότητα. Μέσα στα επόμενα χρόνια προσπαθούσε να βελτιώσει το συγκεκριμένο λογισμικό με κίνητρο να δώσει στο συνθέτη περισσότερη ευελιξία όσο αφορά τα δεδομένα εισόδου και να αξιοποιήσει περισσότερο τις παραμέτρους που αναφέρθηκαν προηγουμένως στο τελικό αποτέλεσμα. Το 1968, είχε αναπτύξει ένα δεύτερο λογισμικό το λεγόμενο Project 2, το οποίο ήταν και πιο περίπλοκο από το προηγούμενο που δημιούργησε. Όπως και στο λογισμικό Project 1, ο συνθέτης είχε την ευκαιρία να δώσει ένα σύνολο από παραμέτρους που θέλει να ληφθούν υπόψη στη σύνθεση που θα επιστρέψει ως έξοδο το λογισμικό. Διάφορες μέθοδοι είχαν αναπτυχθεί στο λογισμικό για την εκτέλεση της δημιουργίας της μελωδίας όπως είναι οι εξής: SEQUENCE, TENDENCY και η GROUP.

Ο Barry Truax είχε δημιουργήσει το 1972-1973 ένα λογισμικό το λεγόμενο POD (Poisson Distribution) στο οποίο η έξοδος που επέστρεφε είχε τη δυνατότητα να συνδέεται αμέσως με συσκευή σύνθεσης ήχου ακούγοντας έτσι τη μελωδία που δημιουργούσε το σύστημα [1]. Κύριο χαρακτηριστικό αυτού του λογισμικού ήταν οι μάσκες τάσης, οι οποίες καθόριζαν τη συχνότητα του χρόνου. Ο συνθέτης δημιουργούσε μοτίβα για τη συχνότητα μπορούσε να

αλλάξει το τέμπο της μελωδία που επιστρεφόταν, αλλά και διάφορες άλλες παραμέτρους της μελωδίας.

Ο Lejaren Hiller το 1955 είχε ως ιδέα η σύνθεση να αναπαριστάται ως μουσικά μηνύματα τα οποία δημιουργούνταν αρχικά από τυχαίες διαδικασίες [1, 7]. Στη συνέχεια το κάθε μουσικό μήνυμα δεχόταν βελτίωση μέσω ενός συνόλου κανόνων, οι οποίοι κανόνες δημιουργούνταν με βάση συγκεκριμένου μουσικού στυλ, έτσι ώστε οι νότες της μελωδίας να μην είναι εντελώς τυχαίες, αλλά να ακολουθούν μια ομοιόμορφη ροή. Στα επόμενα χρόνια ο Lejaren Hiller μαζί με τον Leonard Isaacson είχαν πραγματοποιήσει στο Πανεπιστήμιου Illinois στην Urbana πειράματα για τη δημιουργία παρτιτούρων μέσω των ηλεκτρονικών υπολογιστών όπως φαίνεται και στην επόμενη εικόνα. Συγκεκριμένα είχαν καταφέρει το 1957 να δημιουργήσουν την πρώτη σύνθεση τους, την 'The Illiac Suite for String Quartet' για τέσσερις κινήσεις.



Εικόνα 4: Ο Lejaren Hiller κατά την εκτέλεση των πειραμάτων του.

Σε αυτή τη σύνθεση τους είχαν δημιουργήσει κανόνες οι οποίοι περιείχαν στοιχεία παραδοσιακής μουσικής. Στην πρώτη κίνηση υπήρχε ένα σύνολο από δεκαέξι διαφορετικούς κανόνες οι οποίοι χωρίζονταν σε τρεις κατηγορίες, σε αυτούς που περιείχαν στοιχεία που μπορούσαν να υπάρχουν στη μελωδία, σε αυτούς που περιείχαν στοιχεία που δεν μπορούσαν να χρησιμοποιηθούν στη μελωδία και ως τρίτη κατηγορία ήταν οι κανόνες οι οποίοι περιείχαν ζητούμενα αποτελέσματα. Στη συνέχεια για τη δεύτερη κίνηση είχαν επεκτείνει τα πρώτα τους

πειράματα δίνοντας την ευκαιρία να δημιουργηθούν μελωδίες οι οποίες θα περιέχουν διαφορετικά μουσικά στυλ σε σχέση με το παραδοσιακό που είχαν υιοθετήσει στην πρώτη κίνηση της σύνθεσης τους. Σημαντικό γεγονός ήταν ότι στην τέταρτη κίνηση χρησιμοποιήθηκαν στοχαστικές μέθοδοι όπως είναι οι αλυσίδες Markov οι οποίες στο μέλλον χρησιμοποιήθηκαν σε μελέτες για την αλγοριθμική σύνθεση. Σημαντική επίτευξη του Lejaren Hiller και του Robert Baker ήταν η γλώσσα προγραμματισμού που ανάπτυξαν η λεγόμενη MUSICOMP (Music Simulator Interpret for Compositional Procedures) η οποία περιλάμβανε βοηθητικές λειτουργίες για την αλγοριθμική σύνθεση. Η έξοδος που επέστρεφε το πρόγραμμα μπορούσε είτε να τυπωθεί, είτε να χρησιμοποιηθεί ως είσοδο σε προγράμματα σύνθεσης ήχου. Ο προγραμματιστής επιλέγει ένα σύνολο από στοιχεία τα οποία θα ενισχύουν τους κανόνες που θα τηρεί η μελωδία και στη συνέχεια μέσω διάφορων μεθόδων δημιουργεί το δικό του πρόγραμμα. Μέσω αυτής της γλώσσας προγραμματισμού πολλοί συνθέτες είχαν δημιουργήσει μελωδίες, όπως και ο Hiller ο οποίος είχε δημιουργήσει την παρτιτούρα "Algorithm 1".

1.4 Περιγραφή Προβλήματος

Στόχος της συγκεκριμένης διπλωματικής εργασίας είναι η μελέτη της αλγοριθμικής σύνθεσης αλλά και των γενετικών αλγορίθμων. Προηγούμενη μελέτη στο συγκεκριμένο θέμα έγινε από τον προπτυχιακό φοιτητή του τμήματος Πληροφορικής, Ανδρέα Βανέζη. Ο συγκεκριμένος φοιτητής ανάπτυξε γενετικό αλγόριθμο και κυψελοειδή αυτόματα με σκοπό τη δημιουργία μελωδιών οι οποίες θα έχουν χαρακτηριστικά στοιχεία της jazz μουσικής. Μέσω αυτής της εργασίας και των αποτελεσμάτων της διαπιστώθηκε ότι η χρήση των γενετικών αλγορίθμων δίνει επιθυμητά αποτελέσματα στην αλγοριθμική σύνθεση. Για το λόγο αυτό υπήρξε η επιθυμία και το κίνητρο για δημιουργία προγράμματος το οποίο θα αναπτύξει γενετικό αλγόριθμο με κύριο στόχο τη δημιουργία μελωδιών, οι οποίες θα περιέχουν χαρακτηριστικά στοιχεία της εποχής Μπαρόκ. Ταυτόχρονα, επιθυμούμε οι μελωδίες που θα επιστρέφει το πρόγραμμα να ακούγονται αισθητικά ωραία και θα ικανοποιούν τις επιθυμίες του χρήστη. Η διαφοροποίηση του προγράμματος που δημιουργήθηκε σε σχέση με το αντίστοιχο του Ανδρέα Βανέζη είναι ότι ταυτόχρονα το πρόγραμμα τρέχει το γενετικό αλγόριθμο για δημιουργία πρώτης και δεύτερης φωνής, προσφέρει στο χρήστη τη δυνατότητα να κάνει επιλογές για ορισμένες παραμέτρους του προγράμματος. Συγκεκριμένα ο χρήστης μπορεί να επιλέξει το μουσικό όργανο που επιθυμεί να εκτελέσει τη μελωδία που θα επιστρέψει ο αλγόριθμος και να εισάγει το ποσοστά που επιθυμεί να έχουν τα κριτήρια της συνάρτησης αξιολόγησης. Ο γενετικός αλγόριθμος όπως και οι περισσότεροι αλγόριθμοι θα ξεκινά με ένα τυχαίο αρχικό

πληθυσμό και σταδιακά θα εξελίσσει τον πληθυσμό εκτελώντας μια συνάρτηση ικανότητας, η οποία επιστρέφει κατά πόσο ικανοποιητική είναι η κάθε πιθανή λύση για το συγκεκριμένο πρόβλημα. Στο τέλος, ο αλγόριθμος θα επιστρέφει τη μελωδία η οποία έχει επιτύχει το καλύτερο ποσοστό ικανότητας, δηλαδή είναι η πιο ικανοποιητική μελωδία που δημιουργήθηκε. Η συγκεκριμένη εργασία δίνει τη δυνατότητα στο χρήστη όταν ολοκληρωθεί η εκτέλεση του αλγορίθμου και επιστρέψει τη μελωδία, να ακούσει τη μελωδία αλλά και να κάνει οποιεσδήποτε αλλαγές στις νότες τις μελωδίας που θα επιστρέψει ο αλγόριθμος.

1.5 Δομή Εργασίας

Το υπόλοιπο της εργασίας οργανώνεται σε πέντε κεφαλαία ως εξής: Στο δεύτερο κεφάλαιο θα αναφερθούν διάφορες τεχνικές που δημιουργούν μελωδίες κάνοντας χρήση των ηλεκτρονικών υπολογιστών, καθώς επίσης θα μελετηθούν χαρακτηριστικά στοιχεία της εποχής Μπαρόκ, όπου οι μελωδίες που θα δημιουργούνται θα κληρονομούν στοιχεία αυτής της εποχής.

Στο τρίτο κεφάλαιο θα αναφέρω τις βασικές αρχές των γενετικών αλγορίθμων. Επίσης θα αναφερθούν και άλλες έννοιες τις οποίες θα ήταν καλό να γνωρίζει ο αναγνώστης πριν προχωρήσει στην ανάγνωση της υπόλοιπης διπλωματικής εργασίας. Θα αναφερθεί σε ποιους τομείς χρησιμοποιούνται οι γενετικοί αλγόριθμοι, τα πλεονεκτήματα της χρήσης γενετικών αλγορίθμων. Τέλος, θα συζητήσω τα βήματα που ακολουθούν οι γενετικοί αλγόριθμοι και τις γενετικές λειτουργίες που εκτελούν.

Στο τέταρτο κεφάλαιο θα ξεκινήσει η περιγραφή του κύκλου ανάπτυξης λογισμικού. Πιο συγκεκριμένα θα αναπτυχθεί η φάση απαιτήσεων, προδιαγραφών και λογισμικού.

Στο πέμπτο κεφάλαιο θα συνεχιστεί η περιγραφή των φάσεων του κύκλου ανάπτυξης λογισμικού και συγκεκριμένα θα περιγραφεί η υλοποίηση του προγράμματος που αναπτύχθηκε αναφέροντας τι κάνει η κάθε συνάρτηση στο πρόγραμμα.

Στο τελευταίο κεφάλαιο της διπλωματικής εργασίας θα αξιολογήσω το σύστημα που αναπτύχθηκε με βάση τους αρχικούς στόχους που τέθηκαν στην αρχή και παρατηρήσεις για το σύστημα που αναπτύχθηκε.

Κεφάλαιο 2

Μελωδίες

2.1 Δημιουργία Μελωδιών

Μελωδία είναι μια σειρά από μουσικούς ήχους όπου εκτελώντας την με οποιοδήποτε μουσικό όργανο παράγεται ένα λογικό και συχνά επιθυμητό αποτέλεσμα προς τον ακροατή. Η διαδικασία της παραγωγής μελωδίων έχει εξελιχθεί αρκετά, αφού συνεχώς αναπτύσσονται όλο και περισσότερες μέθοδοι για το συγκεκριμένο θέμα, όπως είναι τα μαθηματικά μοντέλα, συστήματα βασισμένα στην γνώση, γραμματικές και εξελικτικές μέθοδοι.

Όσο αφορά τα μαθηματικά μοντέλα συνήθως δημιουργούν μελωδίες μη-ντετερμινιστικές όπου ο μεταγλωττιστής μέσω από ένα σύνολο γεγονότων προσπαθεί να ελέγξει την πιθανότητα για το κάθε γεγονός και αποφασίζει τον τρόπο με τον οποίο θα παρουσιάσει το αποτέλεσμα έτσι ώστε να ακούγεται μουσικά ενδιαφέρον.

Συγκεκριμένα, ένα μαθηματικό μοντέλο είναι το Markov chains [1, 4, 6, 8] όπου κύριο χαρακτηριστικό είναι ότι μια ή περισσότερες προηγούμενες νότες λαμβάνονται ως είσοδος στο μοντέλο και επιστρέφεται ένα σύνολο πιθανοτήτων για τις επόμενες νότες που μπορεί να ακολουθήσουν. Είναι μια ιδανική μέθοδος για τη δημιουργία μελωδίων που αποτελούνται από συγχορδίες, αλλά όσο αφορά τη δημιουργία μια φωνής για καλύτερα αποτελέσματα είναι προτιμότερο να συνδυαστεί με άλλες μεθόδους. Επιπλέον, μαθηματικά μοντέλα είναι τα χαοτικά συστήματα [8] τα οποία παίρνουν ως είσοδο μια αρχική τιμή και στη συνέχεια εφαρμόζεται σε αυτή μια λειτουργία, η ίδια λειτουργία που εφαρμόζεται στις προηγούμενες εξόδους του μοντέλου. Επαναλαμβάνοντας τη συγκεκριμένη διαδικασία δίνονται ένα σύνολο

από τιμές οι οποίες μπορούν να σχηματίσουν μοτίβα τα οποία μοιάζουν με τις μελωδίες οι οποίες δημιουργούνται από ένα συνθέτη.

Στα συστήματα τα οποία στηρίζονται στη γνώση [8], καθορίζονται από την αρχή ένα σύνολο από κανόνες και περιορισμούς και η έξοδος του συστήματος τηρεί αυτό το σύνολο. Χαρακτηριστικό στοιχείο της συγκεκριμένης μεθόδου είναι ότι για κάθε δράση που γίνεται μπορεί να δώσει εξήγηση για τη συγκεκριμένη επιλογή και ότι πρέπει το σύστημα να είναι προσεκτικό πριν δώσει την τελική λύση.

Συνθέσεις μπορούμε να δημιουργήσουμε χρησιμοποιώντας γραμματικές οι οποίες έχουν την μουσική ως τη γλώσσα της γραμματικής [4, 8]. Οι γραμματικές συνήθως είναι περιοριστικές και περιλαμβάνουν ένα μεγάλο σύνολο από κανόνες και αυτός είναι ο κυρίως λόγος που αρκετοί δεν τις χρησιμοποιούν για παραγωγή μελωδιών. Ωστόσο, ο Steedman [8] ήταν από τους πρώτους ο οποίος ανάπτυξε γραμματική για τη δημιουργία jazz μελωδιών με συγχορδίες, ενώ λίγο αργότερα και συγκεκριμένα το 1991 ο Johnson-Laird [8] είχε δημιουργήσει μια νέα γραμματική για τη δημιουργία jazz μελωδιών με συγχορδίες αλλά και τη δημιουργία μελωδίας για τη φωνή του μπάσου.

Η πιο σημαντική εξελικτική μέθοδος που χρησιμοποιείται τις περισσότερες φορές είναι οι γενετικοί αλγόριθμοι, οι οποίοι δίνουν αρκετά ικανοποιητικά αποτελέσματα [8, 13]. Πρόγραμμα το οποίο κάνει χρήση των γενετικών αλγορίθμων είναι το GenJam [1, 4]. Είναι ένα λογισμικό για σύνθεση μουσικών κομματιών το οποίο χρησιμοποιεί γενετικούς αλγορίθμους και κυρίως χρησιμοποιείται για τη σύνθεση jazz κομματιών. Αρχικά, αρχικοποιεί τον πληθυσμό από τυχαίες νότες και οι δύο πιθανές λύσεις με το μεγαλύτερο ποσοστό ικανότητας αποτελούν τους γονείς. Για να δημιουργηθούν τα παιδιά εκτελείται η λειτουργία της διασταύρωσης (crossover) όπου ανταλλάσσονται τυχαίες νότες μεταξύ των δύο γονέων και ακόμη εκτελείται η λειτουργία μετάλλαξης (mutate) όπου αντιστρέφει νότες σε κάθε γονέα ξεχωριστά. Το διαφορετικό σε αυτό το πρόγραμμα είναι ότι ο χρήστης αξιολογεί τις μελωδίες για τα κριτήρια επιλέγοντας αν είναι καλό ή όχι το κάθε μέρος της μελωδίας, έτσι ώστε να διαδοθούν τα καλά μέρη της μελωδίας στο τελικό αποτέλεσμα. Τα συστήματα τα οποία βασίζονται στη γνώση λαμβάνουν πληροφορίες για τη μουσική από τους χρήστες και από το δημιουργό του συστήματος και μέσω αυτών των πληροφοριών δημιουργούν την τελική τους μελωδία.

Στην συνέχεια ακολουθεί μια αναφορά στις σημαντικότερες μεθόδους που χρησιμοποιούνται για αλγοριθμική σύνθεση.

2.1.1 ΓΕΝΕΤΙΚΟΙ ΑΛΓΟΡΙΘΜΟΙ:

Γενετικοί αλγόριθμοι ανήκουν στην κατηγορία των εξελικτικών αλγορίθμων προσπαθώντας να δώσουν λύσεις σε προβλήματα βελτιστοποίησης [1]. Χρησιμοποιούν διαδικασίες που εμπνέονται από τη φυσική εξέλιξη όπως κληρονομιά, επιλογή και μεταλλαγή. Οι γενετικοί αλγόριθμοι είναι μέθοδοι οι οποίες χρησιμοποιούνται για να αναζητήσουν, αναλύσουν και να λύσουν προβλήματα. Συνδυάζουν σημαντικά στοιχεία από διάφορες τεχνικές αναζήτησης και έχουν ένα σύνολο πιθανών λύσεων. Είναι κατάλληλοι για προβλήματα τα οποία είναι δύσκολο να διαμορφωθούν από μαθηματική άποψη ή δεν έχουν ιδιαίτερο σύστημα κανόνων. Στα βιολογικά μοντέλα οι γενετικοί αλγόριθμοι παρέχουν τους ιδιαίτερους όρους για την επιβίωση και κληρονομικότητα. Τα άτομα και τα χρωμοσώματα που παράγονται αξιολογούνται για τη δυνατότητα τους από τη λειτουργία ικανότητας, η οποία μπορεί να αντιπροσωπεύει μια μαθηματική λειτουργία ή ένα σύστημα κανόνων. Η συνάρτηση αυτή αξιολογεί τις λύσεις σε σχέση με το περιβάλλον στο οποίο βρίσκονται. Οι γενετικοί αλγόριθμοι επιτυγχάνουν επιθυμητά αποτελέσματα στην παραγωγή μουσικών κομματιών και την εξέταση τους αντιπροσωπεύοντας την πολυπλοκότητα των διάφορων πτυχών των μουσικών πληροφοριών. Οι διαδικασίες που μπορεί να χρησιμοποιήσουν οι γενετικοί αλγόριθμοι είναι η διασταύρωση και η μεταλλαγή. Η διαδικασία της αξιολόγησης της ικανότητας δίνει συμπεράσματα για το πόσο επιθυμητά αποτελέσματα έχει το κομμάτι που αναπτύχθηκε.

2.1.2 ΚΥΨΕΛΟΕΙΔΗ ΑΥΤΟΜΑΤΑ:

Τα κυψελοειδή αυτόματα (CA) είναι διακριτά δυναμικά συστήματα τα οποία αναπαριστώνται ως διανύσματα ή πολύ-διαστατούς πίνακες [1, 4, 8]. Για να δημιουργηθεί μια καινούργια γενιά στο κυψελοειδές αυτόματο πρέπει σε όλα τα κελιά του διανύσματος ή του πίνακα να εφαρμοστούν συγκεκριμένοι κανόνες. Η κατάσταση ενός οποιουδήποτε κελιού σε κάποιο συγκεκριμένο χρόνο καθορίζεται σε σχέση με τις τιμές που έχουν τα κελιά που βρίσκονται γειτονικά του σε προηγούμενη στιγμή. Τα κυψελοειδή αυτόματα διακρίνονται σε συγκεκριμένους τύπους ανάλογα με τις διαστάσεις τους. Τα πιο δημοφιλή από αυτά είναι τα 1-διάστασης, 2-διαστάσεων και 3-διαστάσεων κυψελοειδή αυτόματα.

Τα 1-διάστασης κυψελοειδή αυτόματα αποτελούν την πιο απλή μορφή των κυψελοειδών αυτομάτων, όπου αποτελούνται από κύτταρα που έχουν δύο καταστάσεις που

αντιπροσωπεύονται από δυαδικές σειρές συμβόλων και δημιουργούν τη λεγόμενη γειτονιά. Τα κυψελοειδή αυτόματα 1-διάστασης μπορούν να διακριθούν σε τέσσερις άλλες υποκατηγορίες.

- Το αυτόματο συμπεριφέρεται κανονικά και στο τέλος παίρνει μια ομοιογενής μορφή.

- Το αυτόματο παρουσιάζει διαφορετικές τελικές καταστάσεις οι οποίες μπορεί να επαναλαμβάνουν δομές απλά, σταθερά ή περιοδικά .

- Το αυτόματο έχει χαοτική και τυχαία συμπεριφορά.

- Το αυτόματο παρουσιάζει σύνθετα σχέδια τα οποία και μπορούν να επαναληφθούν.

Στα κυψελοειδή αυτόματα 2-διαστάσεων ανήκει και το «Παιχνίδι Της Ζωής», όπου αρχικά τα κύτταρα παίρνουν μια ιδιαίτερη διαμόρφωση αντιπροσωπεύοντας το πρόγραμμα. Αυτά τα αυτόματα ακολουθούν τρεις βασικούς κανόνες. Αρχικά, εάν ένα κενό κύτταρο έχει ακριβώς τρεις γείτονες, ένας μετρητής τοποθετείται στην επόμενη κίνηση. Ο δεύτερος κανόνας λεει ότι κάθε μετρητής με δύο ή τρεις γειτονικούς μετρητές επιζεί για την επόμενη παραγωγή, ενώ ο τελευταίος κανόνας διατυπώνει ότι σε οποιαδήποτε άλλη περίπτωση, κανένας μετρητής δεν τοποθετείται στο κύτταρο.
Στα κυψελοειδή αυτόματα 3-διαστάσεων, η τρίτη διάσταση αντιπροσωπεύει τα βήματα της χρονικής ανάπτυξης. Ένας κανόνας παραγωγής που χρησιμοποιείται για την ανάπτυξη του συγκεκριμένου αυτόματου είναι ότι ένα κύτταρο ενεργοποιείται όταν ακριβώς ένα από τα παρακείμενα κύτταρά του είναι ενεργό, ενώ ήδη τα ενεργά κύτταρα παραμένουν στο ενεργό κράτος τους.
Όσο αφορά την παραγωγή μουσικών κομματιών χρησιμοποιώντας κυψελοειδή αυτόματα, ο Peter Beyls ήταν ο πρώτος που είχε προσπαθήσει να παράξει ένα μουσικό κομμάτι κάνοντας χρήση των κυψελοειδών αυτομάτων. Σε μια συγκεκριμένη εργασία του είχε χρησιμοποιήσει ένα 2-διαστάσεων κυψελοειδές αυτόματο. Αυτό το πρώτο αυτόματο έδινε τη δυνατότητα στον χρήστη να αλλάξει τα αποτελέσματα που έδινε, είτε αλλάζοντας ένα κανόνα παραγωγής, είτε αλλάζοντας την κατάσταση σε ένα κελί. Ο Eduardo Reck Miranda είχε προωθήσει την ιδέα της δημιουργίας ενός μουσικού κομματιού χρησιμοποιώντας δύο κυψελοειδή αυτόματα μέσω της ανάπτυξης του λογισμικού CAMUS, το οποίο έκανε χρήση των Game Of Life και Demon Cyclic Space χρησιμοποιώντας και τα δύο παράλληλα. Στο πρώτο αυτόματο, το Game Of Life

το οποίο δημιουργεί τις νότες στο λογισμικό CAMUS ήταν ένα αυτόματο δύο διαστάσεων το οποίο αποτελείται ένα δυσδιάστατο πίνακα, όπου το κάθε κελί μπορεί να πάρει την τιμή 1 εάν βρίσκεται σε ζωντανή κατάσταση και θα είναι ένα μαύρο κελί ή την τιμή 0 αν βρίσκεται σε κατάσταση νεκρού κελιού και θα έχει λευκό χρώμα. Το κάθε κελί θα βρίσκεται σε συγκεκριμένη κατάσταση σε σχέση με την κατάσταση που βρίσκονται τα οκτώ κελιά που βρίσκονται σε γειτονικές θέσεις από το κελί που εξετάζεται. Οι κανόνες που εφαρμόζονται για να καθοριστεί η κατάσταση των κελιών είναι οι εξής: σε περίπτωση που ένα κελί βρίσκεται σε ζωντανή κατάσταση και το ίδιο ισχύει και για περισσότερους από τέσσερις γείτονες του τότε στην επόμενη στιγμή το συγκεκριμένο κελί θα βρεθεί σε κατάσταση νεκρού. Η δεύτερη περίπτωση είναι όταν ένα κελί είναι ζωντανό αλλά μόνο ένας ή κανένας από τους γείτονες είναι ζωντανός τότε στον επόμενο χρόνο θα πεθάνει το κελί. Ένας ακόμη κανόνας είναι όταν το κελί είναι ζωντανό την προηγούμενη στιγμή και το ίδιο είναι και οι δύο ή τρεις γείτονες του τότε στην επόμενη στιγμή το κελί θα παραμείνει στην ίδια κατάσταση. Η τελευταία περίπτωση είναι όταν το κελί είναι νεκρό και τρεις από τους οκτώ γείτονες του βρίσκονται σε ζωντανή κατάσταση τότε το κελί θα γίνει ζωντανό στην επόμενη του κατάσταση. Όσο αφορά το δεύτερο αυτόματο που χρησιμοποιείται στο σύστημα CAMUS το λεγόμενο Demon Cyclic Space το οποίο και καθορίζει την ενορχήστρωση της σύνθεσης στο συγκεκριμένο σύστημα, αναπαριστάται ως ένα πλέγμα το οποίο σε αυτή την περίπτωση τα κελιά παίρνουν διάφορα χρώματα. Ακόμη μια διαφορά σε σχέση με το αυτόματο Game Of Life είναι ότι τα κελιά που βρίσκονται σε επόμενη χρονική στιγμή καθορίζουν την κατάσταση των κελιών που βρίσκονται σε προηγούμενη στιγμή. Αυτή η ιδιότητα χαρακτηρίζει το αυτόματο ως κυκλικό.

2.1.3 Νευρωνικά δίκτυα:

Ένα νευρωνικό δίκτυο αποτελείται από ένα σύνολο από νευρώνες οι οποίοι ονομάζονται δενδρίτες μέσω των οποίων μεταφέρεται η κίνηση από γειτονικούς νευρώνες στο σώμα των κύτταρων [1, 4, 8]. Τα κύτταρα αντιδρούν σε αυτή την κίνηση μεταφέροντας την κίνηση σε άλλα κύτταρα μέσω του λεγόμενου νευράξονα ο οποίος έχει τη δυνατότητα να επικοινωνεί με τους δενδρίτες, είτε με άλλα κύτταρα. Κάποιος ειδικός μπορεί να αναπτύξει ένα νευρωνικό δίκτυο μαθαίνοντας του να δίνει τα επιθυμητά αποτελέσματα μέσω συγκεκριμένων εισόδων και κανόνων.

2.2 Μουσικά Στοιχεία:

Η μουσική αποτελείται από διάφορα στοιχεία τα οποία χρησιμοποιούνται για να εξυπηρετήσουν τις μουσικές ανάγκες των συνθετών [9, 10]. Στη συνέχεια παρουσιάζονται ορισμένα μουσικά στοιχεία τα οποία θα χρησιμοποιηθούν και στην προγραμματιστική εργασία.

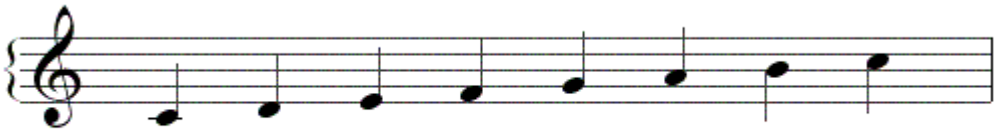
- **Ήχος:**

Ο ήχος αποτελεί οτιδήποτε μπορεί να προκαλέσει τη λειτουργία της ακοής όταν κάποιο αντικείμενο προκαλέσει παλμικές δονήσεις οι οποίες στη συνέχεια θα μεταφερθούν στο τύμπανο του ανθρώπινου αυτιού. Ήχος είναι ακόμη και ο θόρυβος στον οποίο οι παλμικές δονήσεις δημιουργούν ήχους οι οποίοι δεν έχουν συγκεκριμένη οξύτητα σε αντίθεση με τους μουσικούς ήχους οι οποίοι πληρούν τη συγκεκριμένη ιδιότητα. Οι μουσικοί ήχοι χαρακτηρίζονται από το ύψος, την ένταση και την χροιά. Με τον όρο ύψος κάποιου μουσικού ήχου εννοούμε ότι αν ο αριθμός των παλμικών κινήσεων που θα παράγει το μουσικό όργανο ή οποιοδήποτε άλλο αντικείμενο είναι μικρός τότε ο ήχος είναι πιο βαρύς. Στην περίπτωση μεγαλύτερου αριθμού παλμικών κινήσεων ο ήχος είναι οξύς. Όσο αφορά την ένταση του μουσικού ήχου εννοούμε το πόσο δυνατά ή σιγά ακούγεται ένας ήχος. Αν το μουσικό όργανο το οποίο παράγει τον ήχο δεν βρίσκεται σε κοντινή απόσταση από εμάς, εμείς θα άκουμε τον ήχο με μικρότερη ένταση. Η χροιά του ήχου μας δίνει τη δυνατότητα να αντιληφθούμε ότι δύο ήχοι οι οποίοι έχουν το ίδιο ύψος και χροιά εκτελούνται από διαφορετικό αντικείμενο.

- **Κλίμακα:**

Κλίμακα είναι ένα σύνολο από νότες οι οποίες τις περισσότερες φορές είναι συνεχόμενες, αλλά υπάρχουν και οι εξαιρέσεις στις οποίες οι νότες δεν είναι συνεχόμενες. Οι νότες που αποτελούν την κάθε κλίμακα ονομάζονται βαθμίδες και η κάθε κλίμακα ξεκινά από τη νότα που αποτελεί το όνομα της και τελειώνει με την ίδια νότα όμως σε διαφορετικό ύψος. Υπάρχουν διάφορα είδη κλιμάκων αλλά οι πιο συχνά χρησιμοποιούμενες είναι οι μείζονες και οι ελάσσονες οι οποίες αποτελούνται από συνεχόμενες νότες. Η κάθε κλίμακα αποτελείται από ένα σύνολο από διέσεις ή υφέσεις. Το συγκεκριμένο σύνολο λέγεται οπλισμός της κλίμακας και σημειώνεται στην αρχή της μελωδίας. Στην επόμενη εικόνα φαίνεται η Ντο μείζονα

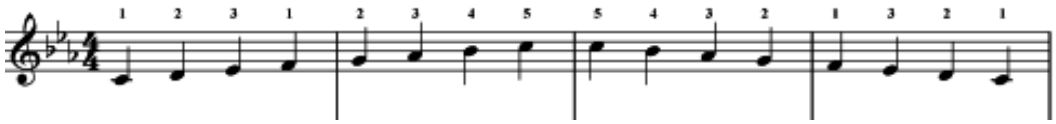
κλίμακα η οποία αποτελείται από συνεχόμενες νότες, όμως δεν υπάρχουν διέσεις αλλά ούτε και υφέσεις.



Εικόνα 5: ΝΤΟ μείζονα

Στην πιο κάτω εικόνα παρουσιάζεται η ΝΤΟ ελάσσονα κλίμακα η οποία αποτελείται από συνεχόμενες νότες και έχει οπλισμό τρεις υφέσεις, όπως φαίνεται στην αρχή της κλίμακας, τη Σι ύφεση, τη ΜΙ ύφεση και τη Λα ύφεση.

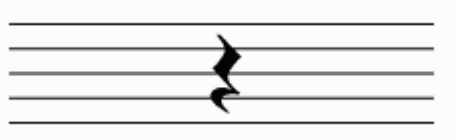
.



Εικόνα 6: ΝΤΟ ελάσσονα

- **Παύσεις:**

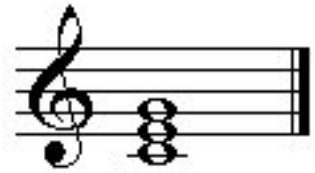
Οι παύσεις υπάρχουν σχεδόν σε όλες τις μελωδίες και είναι συγκεκριμένα σύμβολα όπου όταν τα συναντήσουμε στη μελωδία τότε δεν ακούγεται κάποιος ήχος. Όπως και οι νότες έχουν συγκεκριμένη διάρκεια έτσι και οι παύσεις έχουν τις αντίστοιχες αξίες. Στην παρακάτω εικόνα φαίνεται η παύση τέταρτου, όπου για ένα τέταρτο δεν θα ακούγεται καμία νότα.



Εικόνα 7: Παύση τέταρτου

- **Συγχορδία:**

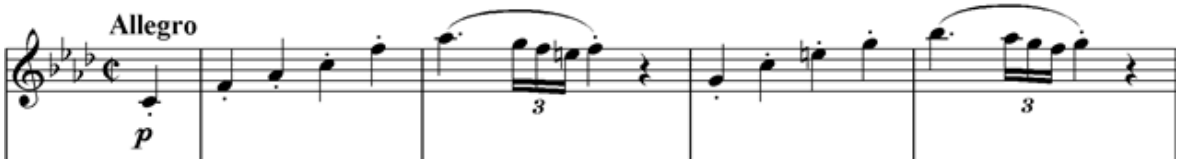
Συγχορδία είναι ένα σύνολο από τουλάχιστον τρεις νότες οι οποίες απέχουν μεταξύ τους απόσταση διαστημάτων $3^{ης}$, οι οποίες ακούγονται ταυτόχρονα. Πιο συχνή χρήση από τους συνθέτες έχουν οι τρίφωνες συγχορδίες, δηλαδή οι συγχορδίες οι οποίες αποτελούνται από τρεις νότες. Η πρώτη νότα της συγχορδίας αποτελεί τη θεμέλιο νότα της συγχορδίας, η επόμενη της νότας απέχει διάστημα $3^{ης}$ από τη θεμέλιο και ονομάζεται τρίτη και στη συνέχεια ακολουθούν οι επόμενες νότες της συγχορδίας. Στο πιο κάτω παράδειγμα παρουσιάζεται μια συγχορδία η οποία έχει ως θεμέλιο νότα την ΝΤΟ νότα, ακολουθεί σε διάστημα $3^{ης}$ από τη θεμέλιο η νότα ΜΙ και μετά είναι η νότα ΣΟΛ η οποία βρίσκεται σε διάστημα $3^{ης}$ από τη ΜΙ νότα.



Εικόνα 8: Συγχορδία ΝΤΟ

- **Μοτίβα:**

Μοτίβο ορίζεται ως ένα σύνολο από ένα μικρό αριθμό από συνεχόμενες νότες οι οποίες προκαλούν τον ακροατή να τις θυμάται εξαιτίας του γεγονότος ότι επαναλαμβάνονται με την ίδια σειρά περισσότερες από μια φορές στη μελωδία. Στα μοτίβα δεν είναι απαραίτητο οι νότες να είναι οι ίδιες, φτάνει οι αξίες των νότων να είναι οι ίδιες. Στην εικόνα 9 παρατηρούμε ότι το μοτίβο το οποίο ξεκινά από το δεύτερο μέτρο και τελειώνει στο τρίτο μέτρο της μελωδίας επαναλαμβάνεται στο τέταρτο και πέμπτο μέτρο. Παρατηρούμε ακόμη ότι το μοτίβο αφορά μόνο τις αξίες των νότων, αφού οι νότες μπορούμε να δούμε ότι αλλάζουν στην επανάληψη του μοτίβου.



Εικόνα 9: Μελωδία η οποία περιέχει μοτίβο

2.3 Εποχή Μπαρόκ:

Η Μπαρόκ μουσική γεννιέται αρχικά στην Ιταλία μεταξύ της περιόδου 1600 και 1750 [11]. Το όνομα της λέγεται ότι το πήρε από τη λέξη μπαρόκο όπου στα πορτογαλικά σημαίνει μαργαριτάρι το οποίο δεν έχει καλό σχήμα. Κατά τη διάρκεια αυτής της περιόδου αναπτύχθηκε το λυρικό θέατρο το οποίο λέγεται και όπερα στην οποία η μελωδία συνοδεύεται από ένα σύνολο από ηθοποιούς οι οποίοι συζητούν μεταξύ τους τραγουδώντας, αλλά και το ορατόριο το οποίο μοιάζει με την όπερα αλλά δεν περιέχει σκηνική δράση. Κατά τη διάρκεια της Μπαρόκ περίοδο δεν εμφανίστηκαν αρκετά νέα μουσικά όργανα, αλλά προσπάθησαν να βελτιώσουν τα παλαιότερα. Μερικά από τα μουσικά όργανα που συνήθιζαν να χρησιμοποιούν κατά τη διάρκεια της συγκεκριμένης περιόδου ήταν το βιολί, η βιόλα, το τσέλο, το πιάνο, το όμποε και το φλάουτο. Οι κύριοι συνθέτες της συγκεκριμένης περιόδου ήταν οι Claudio Monteverdi, Antonio Vivaldi, George Frideric Handel, και ο Johann Sebastian Bach, όπου μετά το θάνατο του τελευταίου επήλθε και το τέλος της μπαρόκ μουσικής.

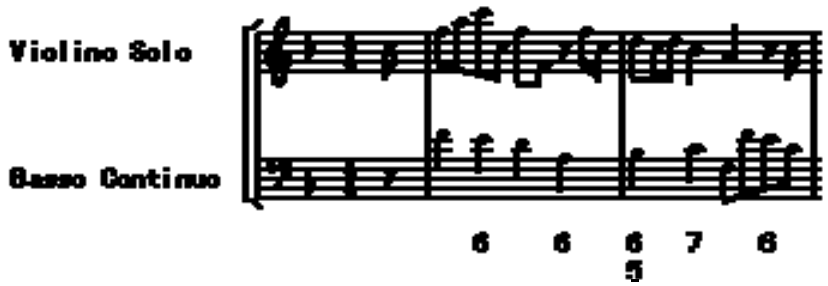
2.4 Χαρακτηριστικά γνωρίσματα της Μπαρόκ εποχής:

Όπως σε κάθε εποχή, έτσι και στη Μπαρόκ οι συνθέσεις που δημιουργήθηκαν περιείχαν συγκεκριμένα χαρακτηριστικά της περιόδου.

- Συγκεκριμένα, για πρώτη φορά χρησιμοποιούνται σειρές από νότες οι οποίες αποτελούν τις μείζονες και ελάσσονες κλίμακες. Οι συνθέτες για να εκφράσουν τη χαρά χρησιμοποιούσαν μείζονες κλίμακες, γρήγορους ρυθμούς και διάφωνα διαστήματα, ενώ για να εκφράσουν τη λύπη χρησιμοποιούσαν ελάσσονες κλίμακες και αργούς ρυθμούς. Στο πρόγραμμα ο χρήστης θα μπορεί να επιλέγει κλίμακα μόνο από ένα σύνολο από μείζονες και ελάσσονες κλίμακες.

- Ακόμη αρκετοί συνθέτες ξεκινούν να χρησιμοποιούν συγχορδίες στις μελωδίες τους. Συχνά οι συνθέτες σημείωναν για την κάθε συγχορδία ένα λατινικό αριθμό, όπως φαίνεται στην εικόνα 10. Η λειτουργία αυτή ονομάζεται συνεχές βάσιμο (basso continuo), όπου και χαρακτηρίζει έντονα αυτή την περίοδο και είναι και ο λόγος που αρκετοί αποκαλούν την

Μπαρόκ εποχή και εποχή συνεχούς βάσιμου. Συνήθως τη μπάσο φωνή έπαιζαν είτε με βιόλα ή με φαγκότο, ένα πνευστό όργανο το οποίο ήταν παρόμοιο με το όμποε. Λόγω του γεγονότος ότι δεν μπορούμε να αναπαραστήσουμε τους λατινικούς αριθμούς που χρησιμοποιούσαν στην εποχή Μπαρόκ εμείς δημιουργούσαμε τις συγχορδίες αναπαριστώντας τις με την πρώτη νότα της κάθε συγχορδίας.



Εικόνα 10: Basso Continuo

- Ακόμη ένα χαρακτηριστικό της συγκεκριμένης περιόδου είναι ότι οι συνθέτες διατηρούσαν το συναίσθημα που επιλέγανε να αναπτύξουν στην αρχή της μελωδίας και διατηρούσαν το ίδιο μέχρι και το τέλος της.

- Παρατηρείται ότι μια μελωδία μπορεί να περιλαμβάνει κάποιο μέρος της το οποίο να επαναλαμβάνεται περισσότερες από μια φορές στην ίδια μελωδία δηλαδή υπάρχουν μοτίβα στη μελωδία το οποία μπορεί να επαναλαμβάνονται σε διαφορετικές θέσεις. Στο πρόγραμμα δημιουργούνται μοτίβα με τις ίδιες αξίες νότων και τα οποία εμφανίζονται σε διαφορετικές θέσεις κάθε φορά, δηλαδή σε διάστημα πέμπτης χαμηλότερα από ότι το προηγούμενο μοτίβο.

- Οι περισσότερες συνθέσεις τις Μπαρόκ εποχής είναι πολύπλοκες στο να εκτελεστούν ή να τραγουδηθούν από κάποιο άτομο. Η συγκεκριμένη μουσική χαρακτηρίζεται για την υπερβολική και περίπλοκη διακόσμηση στην μελωδία.

- Ακόμη οι συνθέτες εκτός από τις συγχορδίες εισήγαγαν και την πολυφωνία στις μελωδίες. Συνήθιζαν να γράφουν στις συνθέσεις τους τη φωνή του μπάσου και της σοπράνο. Στο πρόγραμμα ο γενετικός αλγόριθμος τρέχει για να δημιουργήσει τόσο την πρώτη φωνή όσο και τη δεύτερη φωνή, με αποτέλεσμα στο τέλος να επιστρέφει και τις δύο φωνές.

- Οι μελωδίες που δημιουργούνται τη Μπαρόκ εποχή δεν περιέχουν διακυμάνσεις στην ένταση. Δηλαδή τόσο η έννοια του crescendo (σιγά σε δυνατά) και του decrescendo δυνατά σε σιγά) δεν χρησιμοποιούνται καθόλου. Στο πρόγραμμα που αναπτύχθηκε δεν χρησιμοποιούνται διακυμάνσεις δίνεται στην αρχή της μελωδίας ένας χρωματισμός ο οποίος δεν αλλάζει καθόλου μέχρι το τέλος της μελωδίας.

Κεφάλαιο 3

Γενετικός αλγόριθμος

3.1 Ορισμός Γενετικού αλγορίθμου

Οι γενετικοί αλγόριθμοι είχαν αναπτυχθεί από επιστήμονες της Βιολογίας το 1950, αλλά σημαντική ανάπτυξη είχαν κατά το 1970 από τους John Holland και αρκετούς συνεργάτες του οι οποίοι δημιούργησαν αλγόριθμο χρησιμοποιώντας μηχανισμούς που χρησιμοποιεί η ίδια η φύση για να δώσει λύσεις σε προβλήματα τεχνητής νοημοσύνης [8, 13]. Οι γενετικοί αλγόριθμοι είναι μέθοδοι αναζήτησης οι οποίοι χρησιμοποιούνται για να επιλύσουν προβλήματα αναζήτησης και στόχος τους είναι να δώσουν λύση η οποία θα πληρεί συγκεκριμένα επιθυμητά κριτήρια. Κύριο χαρακτηριστικό των γενετικών αλγόριθμων είναι ότι προσπαθούν να μιμηθούν την ίδια τη φύση μέσω της θεωρίας της εξέλιξης των ειδών η οποία αναπτύχθηκε από τον Δαρβίνο. Χαρακτηριστικά της θεωρίας αυτής που χρησιμοποιούν οι γενετικοί αλγόριθμοι είναι ότι οι οργανισμοί με βάση τις δυσκολίες που συναντούν κατά τη διάρκεια της ζωής του κάποιοι από αυτούς θα επιζήσουν, στη συνέχεια πιθανόν να αλλάξουν και χαρακτηριστικά, ενώ κάποιοι άλλοι θα πεθάνουν. Αλλάζουν τα χρωμοσώματα στους βιολογικούς οργανισμούς, τα οποία είναι μεγάλου μεγέθους μόρια και αποτελούνται από διάφορα στοιχεία που ονομάζονται γονίδια, τα οποία κάποια από αυτά επηρεάζουν συγκεκριμένα χαρακτηριστικά γνωρίσματα του ατόμου. Η πληροφορία η οποία αποθηκεύεται στα γονίδια ονομάζεται γενότυπος. Στη συνέχεια, για να δημιουργηθεί ένας καινούργιος

πληθυσμός αναπαράγονται οργανισμοί οι οποίοι παίρνουν χαρακτηριστικά γονίδια και από τους δύο γονείς.

Έχει παρατηρηθεί ότι οι κλασσικές μέθοδοι αναζήτησης λύσης για ένα πρόβλημα συχνά δεν μπορούν να χρησιμοποιηθούν σε προβλήματα στα οποία το μέγεθος τους είναι αρκετά μεγάλο. Επομένως, υπήρξε το κίνητρο για αναζήτηση νέων μεθόδων επίλυσης προβλημάτων οι οποίες θα χρησιμοποιούσαν την έννοια της πιθανότητας (πιθανοκρατικοί αλγόριθμοι) και θα παρουσίαζαν ικανοποιητικό αποτέλεσμα σε ένα αποδεκτό χρονικό διάστημα. Ένα τέτοιο είδος αλγορίθμου που αναπτύχθηκε ήταν οι γενετικοί αλγόριθμοι οι οποίοι έχουν ως βάση τις αρχές της βιολογικής εξέλιξης και είναι από τις πιο ευέλικτες μεθόδους Οι γενετικοί αλγόριθμοι διατηρούν ένα σύνολο από πιθανές λύσεις του συγκεκριμένου προβλήματος που προσπαθούν να λύσουν και εκτελούν μια αναζήτηση στο χώρο των υποψηφίων λύσεων έχοντας ως κίνητρο να καταλήξουν σε ένα αποδεκτό αποτέλεσμα σε σχέση με συγκεκριμένα κριτήρια. Με αυτή τη μέθοδο σε κάθε επανάληψη δημιουργούνται πληθυσμοί οι οποίοι είναι καλύτεροι προσαρμοσμένοι σε συγκεκριμένο περιβάλλον από τους προηγούμενους μιμούμενοι τους νόμους της φυσικής επιλογής.

Στην αλγοριθμική σύνθεση οι γενετικοί αλγόριθμοι χρησιμοποιούνται αρκετά συχνά. Τρία βασικά συστατικά που πρέπει να καθοριστούν με συνέπεια στην ανάπτυξη του γενετικού αλγορίθμου είναι το πεδίο αναζήτησης, η αναπαράσταση της εισόδου του αλγορίθμου και η συνάρτηση ικανότητας. Όσο αφορά το πεδίο αναζήτησης είναι σημαντικό να περιορίσουμε το πεδίο αναζήτησης εισάγοντας στον αλγόριθμο συγκεκριμένους περιορισμούς οι οποίοι θα εφαρμοστούν για την αναπαράσταση του αρχικού πληθυσμού. Η αναπαράσταση της εισόδου του αλγορίθμου αναπτύσσεται με βάση δύο παράγοντες: τη γνώση, όπου στην αλγοριθμική σύνθεση μπορεί να καθοριστεί ο τόνος, ο ρυθμός, τα μέτρα, κτλ, και το σύνολο των κανόνων οι οποίοι καθορίζουν πώς εξελίσσεται η σύνθεση με βάση τις γενετικές λειτουργίες του αλγορίθμου. Το τελευταίο συστατικό που διαδραματίζει σημαντικό ρόλο στην ανάπτυξη του γενετικού αλγορίθμου είναι η συνάρτηση ικανότητας αφού είναι υπεύθυνη στον καθορισμό του ποια χρωματοσώματα από τον πληθυσμό θα επιζήσουν στην επόμενη γενιά.

3.2 Χρήση Γενετικών Αλγορίθμων

Στη σημερινή εποχή παρατηρείται ότι αρκετά συστήματα αναπτύσσονται χρησιμοποιώντας γενετικούς αλγορίθμους λόγω του γεγονότος ότι προσφέρουν αξιοπιστία στις λύσεις που δίνουν, ευελιξία, αλλά ταυτόχρονα έχουν τη δυνατότητα να συνεργάζονται με άλλες μεθόδους για την επίτευξη κάποιου στόχου [15]. Τομείς όπως τα τεχνητά νευρωνικά δίκτυα, ο

χρονοπρογραμματισμός, τα γραφικά υπολογιστών αλλά και η σύνθεση της μουσικής χρησιμοποιούν ως βάση ανάπτυξης τους γενετικούς αλγορίθμους.

- **Τεχνητά Νευρωνικά Δίκτυα:**

Τα τεχνητά νευρωνικά δίκτυα χρησιμοποιούνται σε τομείς όπως είναι η ιατρική, η επεξεργασία εικόνας και ήχου, η οικονομία και η μηχανολογία. Τα τεχνητά νευρωνικά δίκτυα είναι μοντέλα στα οποία η επεξεργασία τους γίνεται παράλληλα και τα οποία μιμούνται τον τρόπο με τον οποίο οργανώνονται τα νεύρα σε κάθε ανθρώπινο εγκέφαλο. Όσο αφορά τους γενετικούς αλγορίθμους χρησιμοποιούνται έτσι ώστε να δώσουν βέλτιστες λύσεις σε προβλήματα των τεχνητών νευρωνικών δικτύων, αλλά και για να τα εκπαιδεύσουν με αποτέλεσμα να μπορούν να λύσουν προβλήματα τα οποία η πολυπλοκότητα τους είναι αρκετά μεγάλη. Παράδειγμα τέτοιου είδους είναι το σύστημα NeuroGENESYS.

- **Χρονοπρογραμματισμός:**

Με τον όρο χρονοπρογραμματισμός εννοούμε ότι δοθέντος ενός συνόλου πόρων και ενός συνόλου διεργασιών, χρειάζεται να βρεθεί τρόπος να εκτελεστούν οι συγκεκριμένες διεργασίες στον ελάχιστο χρόνο που γίνεται χρησιμοποιώντας κάθε φορά ένα πόρο, στον οποίο πόρο δεν μπορεί ταυτόχρονα να εκτελούνται δύο ή περισσότερες διεργασίες. Σε ορισμένες περιπτώσεις είναι δυνατόν να υπάρξουν και περιορισμοί όσο αφορά συγκεκριμένο χρόνο που πρέπει να ολοκληρωθεί η κάθε διεργασία, τη σειρά με την οποία θα εκτελεστούν οι διεργασίες ή ακόμη και το χρονικό διάστημα που μπορεί να χρησιμοποιηθεί συγκεκριμένος στόχος. Για το λόγο αυτό οι γενετικοί αλγόριθμοι είναι μια αρκετά ικανοποιητική επιλογή για την αντιμετώπιση του προβλήματος του χρονοπρογραμματισμού με τους οποίους μπορεί να εκφραστούν οι συγκεκριμένοι περιορισμοί και να δοθούν επιθυμητά αποτελέσματα μέσω της εκτέλεσης της συνάρτησης ικανότητας η οποία θα αξιολογεί την καταλληλότητα της κάθε διεργασίας για να εκτελεστεί σε κάποιο χρονικό διάστημα.

- **Σύνθεση μουσικής:**

Πολλά προγράμματα συνεχώς δημιουργούνται τα οποία δημιουργούν μουσική συγκεκριμένων ειδών όπως είναι η jazz ή μουσική η οποία περιέχει στοιχεία του στυλ συνθετών. Ένα αρκετά γνωστό πρόγραμμα το οποίο έχει δημιουργηθεί για να μαθαίνει και να παίζει μελωδίες οι οποίες περιέχουν χαρακτηριστικά της jazz μουσικής είναι το GenJam.

- **Γραφικά υπολογιστών:**

Οι γενετικοί αλγόριθμοι χρησιμοποιούνται αρκετά συχνά για τη δημιουργία γραφικών. Συγκεκριμένα, ένα αρκετά γνωστό εργαλείο είναι το genps και το SBαRT, τα οποία δημιουργούν εικόνες κάνοντας χρήση των γενετικών αλγορίθμων.

3.3 Πλεονεκτήματα Γενετικών Αλγορίθμων

Έχουν τη δυνατότητα να επιλύουν δύσκολα προβλήματα γρήγορα, αλλά ταυτόχρονα και αξιόπιστα. Με το πέρασμα των χρόνων έχει παρατηρηθεί ότι σε προβλήματα τα οποία πρέπει να εξεταστούν όλες οι πιθανές λύσεις και να αξιολογηθεί η ορθότητα τους μπορούν να λυθούν πιο αποδοτικά χρησιμοποιώντας γενετικούς αλγορίθμους [14, 15, 16]. Σημειώνεται όμως ότι σε τέτοιου είδους προβλήματα πρέπει από την αρχή να καθοριστεί το πεδίο της έρευνας.

- Οι γενετικοί αλγόριθμοι είναι απλοί στην υλοποίηση τους με αποτέλεσμα να μπορούν να χρησιμοποιηθούν σε ήδη υπάρχοντα συστήματα δίνοντας με αυτό τον τρόπο πιο γρήγορη λύση αλλά ταυτόχρονα και καλύτερη λύση. Δεν χρειάζεται να γνωρίζουν πώς να λυθεί κάποιο πρόβλημα, φθάνει να μπορεί όταν βρεθεί μια ικανοποιητική λύση για το συγκεκριμένο πρόβλημα να μπορεί να την αναγνωρίσει.

- Οι γενετικές λειτουργίες που χρησιμοποιούνται στους συγκεκριμένους αλγόριθμους δεν υλοποιούνται πάντα με τον ίδιο τρόπο με αποτέλεσμα εύκολα να μπορούν να επεκταθούν προσθέτοντας νέα στοιχεία.

- Οι γενετικοί αλγόριθμοι είναι μέθοδοι οι οποίες εξερευνούν το χώρο αναζήτησης τους και ταυτόχρονα χρησιμοποιούν τις επεξεργασμένες τους πληροφορίες.

- Οι υποψήφιες λύσεις στους γενετικούς αλγορίθμους δεν εξαρτώνται η μια από την άλλη με αποτέλεσμα να μπορούν να παραλληλιστούν εκτελώντας κάποιες διαδικασίες τους παράλληλα με αποτέλεσμα να αυξάνεται περισσότερο η αποδοτικότητα τους.

- Έχει παρατηρηθεί ότι οι γενετικοί αλγόριθμοι εφαρμόζονται σε παρά πολλούς τομείς σε σχέση με οποιαδήποτε άλλη μέθοδο εξαιτίας του γεγονότος ότι η επιλογή των κριτηρίων αξιολόγησης γίνεται με βάση το συγκεκριμένο τομέα που επεκτείνεται το πρόβλημα.

- Η συνάρτηση ικανότητας που χρησιμοποιείται στους γενετικούς αλγορίθμους δεν χρειάζεται να περιορίζεται από θορύβους ή οτιδήποτε άλλους περιορισμούς σε σχέση με άλλες μεθόδους οι οποίες πρέπει να τηρούν τέτοιους είδους περιορισμούς. Επίσης, η συνάρτηση ικανότητας είναι το κύριο συστατικό που μελετούν οι γενετικοί αλγόριθμοι κατά τη διαδικασία που ψάχνουν επιθυμητή λύση αξιοποιώντας μόνο όση πληροφορία περιέχεται σε αυτή τη συνάρτηση κάνοντας τον αλγόριθμο πιο ελκυστικό και ευέλικτο.

- Οι γενετικοί αλγόριθμοι έχουν τη δυνατότητα να χρησιμοποιούν πιθανοκρατικούς κανόνες, εφαρμόζοντας το στοιχείο της τύχης στις γενετικές λειτουργίες οι οποίες εκτελούνται σε χρωμοσώματα τα οποία μετά την επιρροή των γενετικών λειτουργιών πάνω τους θα δώσουν καλύτερα αποτελέσματα σε σχέση με άλλα χρωμοσώματα, με αποτέλεσμα να φθάνουμε στην τελική λύση σε πιο γρήγορο χρόνο.

3.4 Διαγραμματική Αναπαράσταση Γενετικού Αλγορίθμου

Οι γενετικοί αλγόριθμοι ακολουθούν ένα σύνολο από συγκεκριμένα βήματα. Αυτά τα βήματα παρουσιάζονται στην εικόνα 14 στην οποία παρουσιάζεται η διαγραμματική αναπαράσταση των γενετικών αλγορίθμων.

Εικόνα 14: Διαγραμματική αναπαράσταση γενετικού αλγορίθμου

3.5 Δομή Γενετικών Αλγορίθμων

Ο γενετικός αλγόριθμος αποτελείται από ένα σύνολο βημάτων τα οποία τηρούνται για να επιτευχθεί η υλοποίηση του [13, 14, 15, 16]. Στη συνέχεια παρουσιάζονται και σχολιάζονται όλα τα βήματα τα οποία και τηρούνται στον γενετικό αλγόριθμο που αναπτύχθηκε.

- **Δημιουργία αρχικού πληθυσμού.**

Στην πρώτη φάση του γενετικού αλγορίθμου δημιουργείται τυχαία ένας αρχικός πληθυσμός ο οποίος αποτελείται από ένα σύνολο κωδικοποιημένων υποψηφίων λύσεων η οποίες ακόμη δεν είναι βέλτιστες και έγκυρες. Η κωδικοποίηση των πιθανών λύσεων γίνεται με μαθηματικό τρόπο, δίνοντας έτσι την ευκαιρία στον ηλεκτρονικό υπολογιστή στη συνέχεια να επεξεργαστεί τις συγκεκριμένες πιθανές λύσεις. Η κωδικοποίηση αυτών των χρωμοσωμάτων συχνά γίνεται είτε με βάση το δυαδικό σύστημα στο οποίο το χρωμόσωμα παρουσιάζεται ως μια δυαδική συμβολοσειρά και επίσης αποτελεί και τον πιο εύκολο τρόπο κωδικοποίησης, είτε με βάση το ακέραιο σύστημα όπου μας δίνει τη δυνατότητα να γλιτώνουμε χρόνο και μνήμη, είτε με οποιοδήποτε άλλο τρόπο επιλέξει ο σχεδιαστής του γενετικού αλγορίθμου. Η διαδικασία της κωδικοποίησης των πιθανών λύσεων πρέπει να γίνει προσεχτικά, έτσι ώστε να αποτελεί την αιτία για την οποία θα αποτύχει ο γενετικός αλγόριθμος.

- **Αντικειμενική συνάρτηση αξιολόγησης.**

Η συνάρτηση αξιολόγησης είναι το πιο σημαντικό μέρος του γενετικού αλγορίθμου. Δέχεται ως είσοδο ένα χρωμόσωμα, δηλαδή μια υποψήφια λύση και την αξιολογεί πόσο κατάλληλη είναι, είτε για να είναι η τελική λύση του προβλήματος, είτε για να συμπεριληφθεί στον καινούργιο πληθυσμό που θα δημιουργηθεί στη συνέχεια, είτε για να αποτελέσει ένα από τους δύο γονείς που θα χρησιμοποιηθούν στις γενετικές λειτουργίες για να δημιουργήσουν τους απογόνους. Αυτή η διαδικασία εκτελείται για όλα τα άτομα του πληθυσμού του προβλήματος. Σημαντικό στοιχείο στη συνάρτηση αξιολόγησης είναι το γεγονός ότι πρέπει ο υπολογισμός της να γίνεται γρήγορα και εύκολα έτσι ώστε ο γενετικός αλγόριθμος να μην καθυστερεί την εκτέλεση του εξαιτίας της.

- **Διαδικασία επιλογής γονέων.**

Σε αυτή τη φάση επιλέγονται τα καταλληλότερα χρωμοσώματα του πληθυσμού, βάση κάποιων κριτηρίων που έχουν καθοριστεί για το πρόβλημα, για να χρησιμοποιηθούν ως γονείς στις διάφορες γενετικές λειτουργίες. Με αυτό τον τρόπο μετά από ορισμένο αριθμό επαναλήψεων του αλγορίθμου τα χαρακτηριστικά στοιχεία των συγκεκριμένων γονέων που επιλέγονται κάθε φορά θα διαδοθούν στους επόμενους πληθυσμούς. Η συγκεκριμένη φάση δεν αλλάζει

χαρακτηριστικά των χρωμοσωμάτων, απλά μόνο επιλέγει κάποια από αυτά για να τροποποιηθούν στη συνέχεια. Μια αρκετά γνωστή μέθοδος επιλογής που χρησιμοποιείται στους γενετικούς αλγόριθμους είναι η λεγόμενη "Επιλογή ρουλέτας". Κάθε χρωμόσωμα έχει πιθανότητα $pi = fi / \sum_{j=1}^{M} fj$ να επιλεγεί όπου fi αναπαριστά πόσο κατάλληλο είναι το κάθε χρωμόσωμα. Η μέθοδος αυτή ονομάζεται 'Επιλογή ρουλέτας' λόγω του γεγονότος ότι μπορούμε να αναπαραστήσουμε το πρόβλημα ως μια ρουλέτα, όπου κάθε χρωμόσωμα κατέχει χώρο στη ρουλέτα όσο είναι και η καταλληλότητα του.

Μια άλλη μέθοδος επιλογής που χρησιμοποιείται συχνά είναι η λεγόμενη 'tournament selection'. Είναι μια αρκετά απλή και ταυτόχρονα γρήγορη μέθοδος η οποία μιμείται την επιλογή που κάνει η φύση για αναπαραγωγή όπου από ένα μικρό σύνολο χρωμοσωμάτων επιλέγει να θέσει για γονέα το χρωμόσωμα το οποίο έχει το μεγαλύτερο ποσοστό ικανότητας από αυτό το σύνολο. Σε αυτή τη μέθοδο δεν χρειάζεται να μελετήσουμε πόσο καλύτερο είναι το ένα χρωμόσωμα από το άλλο, αφού το μόνο που χρειαζόμαστε είναι να επιλέξουμε το καλύτερο χρωμόσωμα από τον πληθυσμό μας.

- **Γενετικές λειτουργίες.**

Οι πιο συνηθισμένες διαδικασίες είναι η διασταύρωση ενός σημείου, διασταύρωση δύο σημείων, ομοιόμορφη διασταύρωση και μετάλλαξη σημείου, οι οποίες αναπτύσσονται στο επόμενο εδάφιο. Σε ένα γενετικό αλγόριθμο δεν είναι απαραίτητο να υπάρχουν όλες οι συγκεκριμένες γενετικές λειτουργίες.

- **Διαδικασία αναπαραγωγής.**

Στη διαδικασία αναπαραγωγής δημιουργούνται δημιουργείται νέος πληθυσμός που αντικαθιστά τον προηγούμενο πληθυσμό με βάση την επίδοση των χρωμοσωμάτων. Το μέγεθος του νέου πληθυσμού πρέπει να παραμείνει το ίδιο με το μέγεθος του αρχικού πληθυσμού. Σε αυτή τη φάση πρέπει να είμαστε προσεχτικοί έτσι ώστε να μην προστεθούν στο νέο πληθυσμό περισσότερα χρωμοσώματα από όσα υπήρχαν στους προηγούμενους πληθυσμούς. Σε περίπτωση που συμβεί αυτό θα πρέπει να αφαιρέσουμε χρωμοσώματα τα οποία έχουν τα μικρότερα ποσοστά ικανότητας. Εξίσου προσεκτικοί πρέπει να είμαστε και στην περίπτωση να δημιουργήσουμε νέο πληθυσμό ο οποίος θα έχει λιγότερα χρωμοσώματα

από τους προηγούμενους. Αν συμβεί κάτι τέτοιο θα πρέπει να προσθέσουμε χρωμοσώματα από τον προηγούμενο πληθυσμό τα οποία δεν υπάρχουν στον νέο πληθυσμό και τα οποία έχουν ικανοποιητικό ποσοστό ικανότητας.

- **Η διαδικασία επαναλαμβάνεται μέχρι να βρεθεί η λύση.**

Ο γενετικός αλγόριθμος τερματίζει όταν βρεθεί μια αποδεκτή λύση με βάση της συνάρτησης καταλληλότητας. Σε διαφορετική περίπτωση, ο αλγόριθμος συνεχίζει να επαναλαμβάνεται.

- **Τερματισμός αλγορίθμου.**

Ο αλγόριθμος τερματίζει σε δύο περιπτώσεις. Η πρώτη περίπτωση είναι όταν έχουμε βρει τη βέλτιστη λύση σύμφωνα με τη συνάρτηση ικανότητας. Μια δεύτερη περίπτωση είναι όταν ο αλγόριθμος έχει επαναληφθεί το μέγιστο αριθμό που έχει αρχικά καθοριστεί είτε τυχαία είτε από το χρήστη. Σε αυτή την περίπτωση συνηθίζεται να τερματίζει ο αλγόριθμος ‘σκόπιμα’ επειδή ο πληθυσμός μπορεί να μη δέχεται επιπλέον βελτιστοποιήσεις με αποτέλεσμα να τον τερματίζουμε, να ελέγχουμε τις λύσεις που πήραμε μέχρι στιγμής και αν δεν είμαστε ικανοποιημένοι από τα υπάρχων αποτελέσματα ξεκινάμε τον αλγόριθμο από την αρχή για να πάρουμε καινούργια και ίσως καλύτερα αποτελέσματα.

3.6 Γενετικές Λειτουργίες

Οι βασικές λειτουργίες που χρησιμοποιούνται στους γενετικούς αλγορίθμους είναι η διασταύρωση και η μετάλλαξη. Η διασταύρωση έχει την ιδιότητα να δημιουργεί συγκεκριμένο αριθμό από καινούργια χρωμοσώματα τα οποία υιοθετούν συγκεκριμένο γενετικό υλικό από τους γονείς. Τα υπόλοιπα χρωμοσώματα παραμένουν όπως είναι για να δημιουργήσουν όλα μαζί καινούργιο πληθυσμό. Όσο αφορά τη διαδικασία της μετάλλαξης έχει τη δυνατότητα να ανακτήσει χαμένη πληροφορία σε κάποιο γονίδιο. Επομένως, η μετάλλαξη προκαλεί μικρές αλλαγές στα χρωμοσώματα σε σχέση με τους γονείς.

- **Διασταύρωση ενός σημείου – single point crossover:**

Η συγκεκριμένη διαδικασία είναι μια αρκετά απλή μέθοδος η οποία χρησιμοποιείται αρκετά συχνά στους γενετικούς αλγόριθμους. Σε αυτή τη διαδικασία επιλέγεται τυχαίο σημείο στα χρωμοσώματα το οποίο μπορεί να πάρει τιμές από 1 μέχρι και το μήκος του χρωμοσώματος μείον ένα. Ο συγκεκριμένος αριθμός δίνει τη θέση από την οποία θα ξεκινήσει να γίνει η ανταλλαγή στοιχείων μεταξύ των δύο γονέων για να δημιουργηθούν οι καινούργιοι απόγονοι.

Παραδείγματος χάριν, έχουμε τους εξής δύο γονείς:

x1 = 11110010

x2 = 10110100

αν το σημείο που επιλέχθηκε για να γίνει η ανταλλαγή είναι 4 τότε οι δύο νέοι απόγονοι που θα δημιουργηθούν θα είναι οι εξής:

x11 = 11110100

x22 = 10110010

- **Διασταύρωση δύο σημείων:**

Στη λειτουργία της διασταύρωσης δύο σημείων η ανταλλαγή γενετικού υλικού γίνεται με βάση δύο σημεία. Το πρώτο σημείο αντιπροσωπεύει τη θέση που θα ξεκινήσει να γίνεται ανταλλαγή και το δεύτερο σημείο αντιπροσωπεύει την θέση στην οποία θα τελειώσει να γίνεται ανταλλαγή.

Παραδείγματος χάριν, έχουμε τους εξής δύο γονείς:

x1 = 11010010

x2 = 10100100

και τα δύο σημεία που επιλέχθηκαν για να γίνουν ανταλλαγές είναι 2 και 4. Τότε από τους δύο γονείς θα προκύψουν οι δύο απόγονοι:

x11 = 10100010

x22 = 11010100

- **Ομοιόμορφη διασταύρωση:**

Σε αυτή τη μέθοδο έχουμε μια πιθανότητα όπου ανάλογα με αυτήν επιλέγεται από ποιο γονέα το νέο χρωμόσωμα θα πάρει το γονίδιο. Η διαδικασία αυτή επαναλαμβάνεται για όλα τα γονίδια των γονέων με αποτέλεσμα να απαιτείται οι δύο γονείς να έχουν το ίδιο μέγεθος.

- **Αριθμητική διασταύρωση:**

Στην αριθμητική διασταύρωση οι δύο γονείς δέχονται μια αριθμητική πράξη και δίνεται ως νέος απόγονος το αποτέλεσμα που λαμβάνεται από τη συγκεκριμένη αριθμητική πράξη.

- **Μετάλλαξη σημείου:**

Η λειτουργία της μετάλλαξης επιλέγεται ένα τυχαίο γονίδιο να αλλάξει τιμή με βάση κάποιο συγκεκριμένο κανόνα που τίθεται στον αλγόριθμο, ενώ τα υπόλοιπα γονίδια παραμένουν με τις ίδιες τιμές. Το κάθε γονίδιο έχει την ίδια πιθανότητα να επιλεγεί για να μεταλλαχθεί. Με αυτή τη διαδικασία σε περίπτωση που βρισκόμαστε σε ένα αρκετά κοντινό σημείο σε σχέση με τη βέλτιστη λύση που θέλουμε να επιτύχουμε, η λειτουργία της μετάλλαξης θα μας βοηθήσει.

Παραδείγματος χάριν, αν έχουμε τον εξής γονέα:

x1 = 10101011

και επιλέγεται να αλλάξει η τιμή στη θέση 2, τότε ο νέος απόγονος που θα προκύψει θα είναι ο εξής:

x11 = 11101011

- **Εναλλαγή:**

Η γενετική λειτουργία της εναλλαγής επεξεργάζεται ένα χρωμόσωμα και δίνει ως αποτέλεσμα ένα καινούργιο χρωμόσωμα. Βασικά επιλέγονται δύο διαφορετικά σημεία στο αρχικό χρωμόσωμα και στη συνέχεια αλλάζει τη σειρά των γονιδίων που βρίσκονται μεταξύ των δύο σημείων.

- **Σύνταξη:**

Η σύνταξη λαμβάνει ένα χρωμόσωμα και εφαρμόζει σε αυτό το χρωμόσωμα συγκεκριμένους κανόνες που έχουν καθοριστεί στην αρχή, με αποτέλεσμα να επιστρέφει ένα απόγονο ο οποίος είναι σε πιο απλή μορφή σε σχέση με το γονέα του.

Κεφάλαιο 4

Ανάπτυξη συστήματος

4.1 Ορισμός Απαιτήσεων (requirements):

Στόχος της εργασίας είναι η δημιουργία λογισμικού το οποίο θα χρησιμοποιεί γενετικό αλγόριθμο για τη δημιουργία μελωδιών οι οποίες θα περιλαμβάνουν στοιχεία από την εποχή Μπαρόκ. Η φάση των απαιτήσεων καθορίζει το βασικό σκοπό της εργασίας.

4.1.1 Σύνθεση:

Το σύστημα δέχεται ως είσοδο την αρχική κλίμακα, τον αριθμό των μέτρων και στη συνέχεια το πρόγραμμα θα επιστρέφει μια μελωδία. Η ποιότητα της μελωδίας που θα δημιουργηθεί πρέπει να είναι όσο το δυνατό καλύτερη τη δεδομένη στιγμή. Η μελωδία θα συνοδεύεται από τη δεύτερη φωνή αλλά και από τις συγχορδίες. Η πρώτη όπως και η δεύτερη φωνή δημιουργούνται τυχαία, ενώ οι συγχορδίες έχουν ως πρώτη νότα τη νότα που υπάρχει στην ίδια θέση στη μελωδία της πρώτης φωνής. Στο τέλος το σύστημα πρέπει να μπορεί να εκτελέσει τη μελωδία που θα επιστρέψει ο γενετικός αλγόριθμος και ακόμη πρέπει να έχει τη δυνατότητα να αποθηκεύσει τη συγκεκριμένη μελωδία.

4.1.2 Γενετικός αλγόριθμος:

Ο γενετικός αλγόριθμος πρώτα θα δημιουργεί τυχαίες μελωδίες οι οποίες θα αποτελούνται από τυχαίες νότες και αξίες και θα αποτελούν την πρώτη φωνή και μέσω των οποίων θα δημιουργούνται και οι συγχορδίες. Στη συνέχεια θα δημιουργείται για κάθε μία από αυτές τις μελωδίες μια δεύτερη μελωδία η οποία θα αποτελεί τη δεύτερη φωνή και στο τέλος οι συγκεκριμένες μελωδίες θα καταλήξουν να περιλαμβάνουν νότες οι οποίες απέχουν από τις αντίστοιχες νότες της πρώτης φωνής διαστήματα τρίτης, πέμπτης, ογδόης ή να είναι οι ίδιες νότες.

Σε κάθε μελωδία θα εκτελείται ένα σύνολο από γενετικές λειτουργίες όπως είναι η διασταύρωση και η μετάλλαξη. Οι γονείς που επιλέγονται για να χρησιμοποιηθούν στις γενετικές λειτουργίες είναι οι δύο μελωδίες με τα ψηλότερα ποσοστά ικανότητας. Η διασταύρωση γίνεται από ένα τυχαίο μέτρο μέχρι το τέλος της μελωδίας και στη μετάλλαξη επιλέγεται μια τυχαία νότα η οποία τηρεί συγκεκριμένους περιορισμούς για να αλλαχθεί και να πάρει τη θέση της μια καλύτερη νότα.

Η ικανότητα της κάθε μελωδίας θα καθορίζεται με βάση του αθροίσματος του ποσοστού που ικανοποιείται σε κάθε κριτήριο. Για κάθε κριτήριο ο χρήστης μπορεί να αναθέσει ένα συγκεκριμένο βάρος το οποίο θα λαμβάνει τιμή από 0 μέχρι και 1, το οποίο θα πολλαπλασιαστεί στη συνέχεια με τον αριθμό των νότων που ικανοποιούν το συγκεκριμένο κριτήριο. Το τελικό ποσοστό ικανότητας της κάθε μελωδίας πρέπει παίρνει τιμή από 0 μέχρι και τη μέγιστη τιμή που ζήτησε στην αρχή του αλγορίθμου ο χρήστης.

Τα κριτήρια τα οποία χρησιμοποιούνται στο γενετικό αλγόριθμο υπάρχουν για να αξιολογούν τα παρακάτω στοιχεία της κάθε μελωδίας:

- Αν οι νότες ανήκουν στην αρχική κλίμακα.
- Αν υπάρχουν επιτρεπτά και μη επιτρεπτά διαστήματα μεταξύ συνεχόμενων νότων στη μελωδία.
- Επανάληψη ίδιων νότων συνεχόμενα.
- Ύπαρξη μοτίβων στη μελωδία.
- Συνέπεια μεταξύ πρώτης και δεύτερης φωνής.

Η επιλογή του νέου πληθυσμού γίνεται επιλέγοντας τις μελωδίες που υπάρχουν στον τελευταίο πληθυσμό οι οποίες φέρουν τα καλύτερα ποσοστά ικανότητας.

Ο γενετικός αλγόριθμος πρέπει να τερματίζει μόνο αν επαναλήφθηκε τις μέγιστες φορές που ζήτησε ο χρήστης ή σε περίπτωση που εντοπίστηκε μελωδία με το μέγιστο ποσοστό ικανότητας που ζήτησε ο χρήστης.

4.1.3 Διασύνδεση Με Τον Χρήστη:

Το σύστημα πρέπει να παρουσιάζει στο χρήστη μια φόρμα μέσω της οποίας θα έχει τη δυνατότητα να δίνει συγκεκριμένες παραμέτρους για είσοδο στο γενετικό αλγόριθμο. Ο χρήστης μπορεί να τροποποιήσει παραμέτρους του γενετικού αλγορίθμου πριν την εκτέλεση του.

Γενικές Επιλογές: αφορούν παραμέτρους που αφορούν το μέγεθος του πληθυσμού και το μέγιστο αριθμό επαναλήψεων.

Αρχικοποίηση: είναι παράμετροι για την αρχική κλίμακα της μελωδίας και τον αριθμό των μέτρων της μελωδίας.

Συνάρτηση Ικανότητας: ο χρήστης έχει τη δυνατότητα να καθορίσει το βάρος του κάθε κριτηρίου.

Μουσικά Όργανα: ο χρήστης επιλέγει ποιο όργανο επιθυμεί να εκτελέσει την τελική μελωδία.

Η κάθε μελωδία που επιστρέφεται έχει τη δυνατότητα να φυλάγεται και να μπορούμε να την ακούσουμε οποιαδήποτε στιγμή.

4.2 Ορισμός Προδιαγραφών

Για την υλοποίηση του γενετικού αλγορίθμου θα εκτελεστούν κάποια συγκεκριμένα βήματα. Πρώτα από όλα θα γίνεται αρχικοποίηση του γενετικού αλγορίθμου, στη συνέχεια θα αξιολογούνται οι μελωδίες που δημιουργήθηκαν, θα εκτελούνται γενετικές λειτουργίες στις μελωδίες και ο αλγόριθμος θα τερματίζει όταν θα πάρουμε επιθυμητό αποτέλεσμα, είτε αν το επιθυμεί ο ίδιος ο χρήστης. Η συγκεκριμένη διαδικασία περιγράφεται παρακάτω:

Αρχικοποίηση:

Αρχικά, θα πρέπει να επιλεγεί ένας πληθυσμός μελωδιών οι οποίες θα αρχικοποιούν τον αλγόριθμο. Η κάθε μελωδία από αυτές θα πρέπει να ικανοποιεί ένα σύνολο από κανόνες οι οποίοι είναι οι εξής: κάθε μελωδία θα αποτελείται από ένα αριθμό από μέτρα και θα έχει ως πρώτη νότα τη θεμέλιο νότα της κλίμακας που θα επιλέξει ο χρήστης για να δημιουργηθεί η μελωδία και θα πρέπει να τελειώνει με την ίδια νότα. Η κάθε μελωδία θα τελειώνει με συγκεκριμένη πτώση, δηλαδή οι τελευταίες νότες της κάθε μελωδίας θα είναι η τέταρτη νότα της κλίμακας, η θεμέλιος, θα ακολουθήσει η δεσπόζουσα και ως τελευταία νότα θα είναι η θεμέλιος νότας της κλίμακας. Οι διάρκειες των νότων θα είναι αξίας μισού, τέταρτου, ογδόου και δέκατου έκτου. Στην περίπτωση που υπάρχει δέκατο-έκτο στη συνέχεια θα ακολουθήσει ακόμη μια νότα με την ίδια αξία. Θα επιτρέπονται να χρησιμοποιηθούν παύσεις των αντίστοιχων αξίων. Όταν δημιουργηθεί η πρώτη φωνή τότε με βάση τη νότα που υπάρχει σε κάθε θέση δημιουργείται και η αντίστοιχη συγχορδία. Δηλαδή αν η νότα στην πρώτη φωνή είναι ΝΤΟ τότε θα δημιουργηθεί η συγχορδία ΝΤΟ-ΜΙ-ΣΟΛ. Η συγκεκριμένη συγχορδία έχει την ίδια αξία με τη νότα της πρώτης φωνής. Στη συνέχεια θα επαναλαμβάνεται η ίδια διαδικασία για να δημιουργηθούν μελωδίες οι οποίες θα αποτελούν τη δεύτερη φωνή. Σε αυτή την περίπτωση ελέγχεται η αξία της κάθε νότας που βρίσκεται στην πρώτη φωνή. Αν στην πρώτη φωνή υπάρχει μισό στη δεύτερη φωνή θα υπάρχουν δύο τέταρτα, αν υπάρχει τέταρτο στην πρώτη φωνή τότε στη δεύτερη φωνή θα υπάρχουν δύο όγδοα, αν υπάρχει όγδοο θα ακολουθήσουν στη δεύτερη φωνή δύο δέκατα-όγδοα, σε περίπτωση δύο δέκατων-έκτων στην πρώτη περίπτωση θα υπάρξει ένα όγδοο γιατί στην πρώτη φωνή θα ακολουθήσει ακόμα ένα δέκατο-έκτο.

Συνάρτηση ικανότητας:

Το σύνολο των ατόμων που αποτελούν τον πληθυσμό του αλγορίθμου θα αξιολογείται με βάση τη συνάρτηση ικανότητας, η οποία περιλαμβάνει ένα σύνολο κριτηρίων που θα πρέπει να ικανοποιεί στο σύνολο κάποιο ποσοστό. Η συνάρτηση ικανότητας θα υπολογίζεται πολλαπλασιάζοντας το βάρος του κάθε κριτηρίου, το οποίο μπορεί να καθορίσει στην αρχή του αλγορίθμου ο χρήστης, επί τον αριθμό των νότων που ικανοποιούν το συγκεκριμένο κριτήριο.

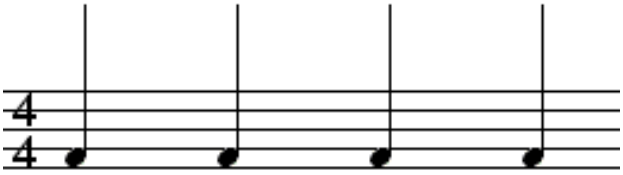
Συνάρτηση:

$$X = \sum_{i=1}^{n} w_i x_i \qquad \text{όπου } x_i \in [0,100] \text{ και } w_i \in [0,1]$$

Κριτήρια:

Τα κριτήρια που θα αποτελούν τη συνάρτηση ικανότητας είναι τα εξής:

- **Τονικότητα:** κάθε φορά γίνεται έλεγχος για όλες τις νότες που αποτελούν τις μελωδίες αν αποτελούν νότες της αρχικής κλίμακας που επίλεξε ο χρήστης. Όσες περισσότερες νότες που ανήκουν στην αρχική κλίμακα μας περιέχονται στη μελωδία τόσο προτιμότερο είναι για τον αλγόριθμο.
- **Διαστήματα οκτάβας:** κάθε φορά γίνεται έλεγχος κατά πόσο υπάρχουν συνεχόμενες νότες στις μελωδίες οι οποίες απέχουν μεταξύ τους διαστήματα οκτάβας. Μελωδίες οι οποίες περιέχουν νότες οι οποίες απέχουν διάστημα οκτάβας είναι καλύτερες, γιατί τα διαστήματα οκτάβας αποτελούν χαρακτηριστικό στοιχείο της Μπαρόκ εποχής.
- **Διαστήματα πέραν της οκτάβας:** σε όλες τις μελωδίες ψάχνουμε αν υπάρχουν συνεχόμενες νότες οι οποίες βρίσκονται σε μεταξύ τους απόσταση πέραν της μιας οκτάβας. Όταν υπάρχουν τέτοιου είδους διαστήματα επειδή δεν είναι επιθυμητά, πρέπει να αποφεύγονται να επιλέγονται μελωδίες οι οποίες περιέχουν συγκεκριμένα διαστήματα.
- **Μοτίβα:** θα γίνεται έλεγχος σε κάθε μελωδία κατά πόσο περιέχονται μοτίβα. Τα μοτίβα αποτελούν χαρακτηριστικό γνώρισμα της Μπαρόκ εποχής γι' αυτό μελωδίες οι οποίες περιέχουν μοτίβα προτιμώνται σε σχέση με άλλες οι οποίες δεν περιέχουν.
- **Συνεχόμενες νότες:** ελέγχεται κατά πόσο σε μια μελωδία υπάρχουν τρεις ή περισσότερες φορές συνεχόμενα ίδιες νότες. Σε μια μελωδία δεν είναι επιθυμητό να υπάρχουν συνεχώς οι ίδιες νότες για αυτό το λόγο αποφεύγονται μελωδίες οι οποίες περιέχουν το συγκεκριμένο μειονέκτημα.



Εικόνα 15: Μέτρο το οποίο περιέχει συνεχόμενες ίδιες νότες

- **Συνεχόμενες παύσεις:** ελέγχουμε σε όλες τις μελωδίες αν υπάρχουν τρεις ή περισσότερες συνεχόμενες παύσεις. Δεν επιθυμούμε οι μελωδίες μας να περιέχουν για αρκετό χρονικό διάστημα παύσεις γιατί με αυτόν τον τρόπο δεν υπάρχει αρμονία στη μελωδία μας και θα υπάρχει σημείο στο οποίο δεν θα ακούγεται οτιδήποτε στη μελωδία μας και έτσι προσπαθούμε να αποφεύγουμε μελωδίες με συνεχόμενες παύσεις.

- **Συσχετιζόμενες νότες μεταξύ πρώτης και δεύτερης μελωδίας:** αντιστοιχίζουμε τις νότες της πρώτης μελωδίας με της δεύτερης μελωδίας για να δούμε κατά πόσο οι νότες της δεύτερης μελωδίας απέχουν από τις νότες της πρώτης μελωδίας διαστήματα τρίτης, πέμπτης, όγδοης ή είναι οι ίδιες νότες.

Επιλογή:

Στη συνέχεια αφού εκτελεστεί η συνάρτηση ικανότητας επιλέγεται ένα σύνολο ατόμων από τον πληθυσμό ανάλογα με το ποσοστό αξιολόγησης τους για να δημιουργήσουν τη νέα γενεά. Στη συγκεκριμένη περίπτωση επιλέγονται οι μελωδίες οι οποίες έχουν τα καλύτερα ποσοστά ικανότητας για να αποτελέσουν το νέο πληθυσμό. Αυτό γίνεται έτσι ώστε να συνεχίσουν να επιζούν οι μελωδίες οι οποίες έχουν τα καλύτερα αποτελέσματα και με αυτό τον τρόπο να προκύψει γρηγορότερα η επιθυμητή μελωδία. Ο αριθμός των ατόμων που αποτελούν τον αρχικό πληθυσμό είναι ο ίδιος με τον αριθμό των ατόμων που αποτελούν τον καινούριο πληθυσμό και τον οποίο έχει επιλέξει ο χρήστης στην αρχή του αλγόριθμου.

Γενετικές λειτουργίες:

Θα ακολουθήσει η επιλογή δύο μελωδιών οι οποίες θα αποτελούν τους γονείς του πληθυσμού και θα χρησιμοποιηθούν στις γενετικές λειτουργίες. Οι δύο γονείς που θα επιλεγούν θα είναι αυτοί οι οποίοι μέχρι στιγμής έχουν επιτύχει το μεγαλύτερο ποσοστό ικανότητας. Οι βασικές γενετικές λειτουργίες που θα χρησιμοποιηθούν στο πρόγραμμα θα είναι οι εξής:

- Διασταύρωση ενός σημείου: για να δημιουργηθεί μια καινούργια μελωδία παίρνει ένα μέρος από τη μελωδία του πρώτου γονέα και το υπόλοιπο το παίρνει από το δεύτερο γονέα.
- Διασταύρωση δύο σημείων: για να δημιουργηθεί μια νέα μελωδία παίρνει την αρχή και το τέλος από τη μελωδία του πρώτου γονέα και το υπόλοιπο μέρος το παίρνει από το δεύτερο γονέα.
- Μετάλλαξη σημείου: κάθε φορά θα επιλέγεται τυχαία μια νότα της μελωδίας και θα ελέγχεται αν είναι η πρώτη ή τελευταία νότα της μελωδίας. Αν είναι μια από τις δύο τότε ελέγχουμε αν είναι η θεμέλιος νότα της κλίμακας. Αν ναι, δεν γίνεται καμία αλλαγή, αλλιώς μετατρέπουμε τη συγκεκριμένη νότα στη θεμέλιο. Αν είναι οποιαδήποτε άλλη νότα τότε ελέγχουμε αν η νότα ανήκει στην αρχική κλίμακα. Αν ναι, δεν γίνεται κάποια αλλαγή διαφορετικά επιλέγουμε τυχαία μια νότα της αρχικής κλίμακας.

Εφόσον δεν παίρνουμε το επιθυμητό αποτέλεσμα ο αλγόριθμος επαναλαμβάνεται για να δεχτεί η μελωδία περαιτέρω βελτιστοποιήσεις μέχρι να πάρουμε το επιθυμητό αποτέλεσμα.

Συνθήκες τερματισμού:

Το πρόγραμμα θα τερματίζει όταν ικανοποιηθεί κάποια από τις συνθήκες τερματισμού οι οποίες είναι οι ακόλουθες:

- Η μελωδία να έχει αξιολογηθεί από τη συνθήκη ικανότητας δίνοντας ένα ικανοποιητικό ποσοστό το οποίο ζήτησε στην αρχή ο χρήστης.
- Ο αλγόριθμος έχει επαναληφθεί σε ένα μέγιστο αριθμό, ο οποίος δίνεται από το χρήστη.

Με το που θα τερματίσει ο γενετικός αλγόριθμος το πρόγραμμα θα επιστρέψει την καλύτερη μελωδία που προέκυψε μέχρι στιγμής και ο χρήστης θα έχει τη δυνατότητα αν θέλει να την ακούσει και να την αποθηκεύσει.

Περιβάλλον ανάπτυξης λογισμικού

Θέλουμε να χρησιμοποιήσουμε μια γλώσσα προγραμματισμού για την ανάπτυξη γενετικού αλγορίθμου η οποία θα μας δίνει αποτελέσματα σε γρήγορο χρόνο. Για το λόγο αυτό θα χρησιμοποιηθεί η γλώσσα προγραμματισμού Java η οποία είναι μια δυναμική γλώσσα και προσφέρει αυτόματη διαχείριση μνήμης και υψηλού επιπέδου δομές δεδομένων. Συγκεκριμένα

από τη γλώσσα προγραμματισμού Java χρησιμοποιώ τη βιβλιοθήκη της JMusic, η οποία προσφέρει στο χρήστη τη δυνατότητα να δημιουργήσει μελωδίες, τις οποίες τις μετατρέπει σε μορφή Standard MIDI File, αρχείο το οποίο ο χρήστης μπορεί να ακούσει ή μπορεί να γραφτεί η μελωδία σε CPN(common practise notation).

Παράμετροι εισόδου:

Ο χρήστης θα έχει τη δυνατότητα να δίνει κάποιες παραμέτρους ως είσοδο στο πρόγραμμα. Οι παράμετροι θα είναι οι εξής:

- Την κλίμακα στην οποία θα ανήκει το μουσικό κομμάτι.
- Μέγιστος αριθμός μέτρων που θα αποτελείται η μελωδία.
- Μέγιστος αριθμός πληθυσμού του αλγορίθμου.
- Μέγιστος αριθμός που θα επαναληφθεί ο αλγόριθμος.
- Επιλογή γενετικής λειτουργίας που θέλει να εκτελεστεί.
- Ποια κριτήρια επιθυμεί να χρησιμοποιούνται στο πρόγραμμα.

4.3 ΦΑΣΗ ΣΧΕΔΙΑΣΜΟΥ:

- **Λειτουργίες σχετιζόμενες με τον χρήστη:**

(i) Σκοπός:

Ο χρήστης να μπορεί να δώσει στην αρχή του προγράμματος κάποιες επιλογές που θέλει να συμπεριληφθούν στη μελωδία που θα δημιουργηθεί. Χαρακτηριστικά ο χρήστης θα μπορεί να επιλέξει σε ποια κλίμακα θέλει να δημιουργηθεί το μουσικό κομμάτι από ένα συγκεκριμένο σύνολο μειζόνων και ελασσόνων κλιμάκων που του παρουσιάζονται. Θα έχει τη δυνατότητα να επιλέξει τον αριθμό μέτρων που θα περιέχει η μελωδία, το μέγιστο αριθμό πληθυσμού του αλγόριθμου ο οποίος θα πρέπει να κυμαίνεται μεταξύ 1 και 200, το ποσοστό που θέλει να ικανοποιεί η συνάρτηση ικανότητας για να θεωρηθεί επιθυμητό το αποτέλεσμα της μελωδίας και τον αριθμό που θα θέλει να επαναληφθεί ο αλγόριθμος μέχρι να δώσει το μουσικό κομμάτι, ο οποίος αριθμός δεν μπορεί να είναι μικρότερος του 0 και μεγαλύτερος του 200, το βάρος που επιθυμεί να έχει το κάθε κριτήριο από μια λίστα που θα του δίνεται και ακόμη το

μουσικό όργανο για το οποίο επιθυμεί να εκτελεστεί η τελική μελωδία στο τέλος του αλγορίθμου μέσω ενός συνόλου μουσικών οργάνων που θα του παρουσιάζεται.

- **<u>Λειτουργίες οι οποίες θα εκτελεί η αλγόριθμος:</u>**

Αφού έχουν εκχωρηθεί στο πρόγραμμα οι απαραίτητες παράμετροι, θα ξεκινήσει η εκτέλεση του αλγορίθμου.

Αρχικά, δημιουργείται ένα σύνολο n μελωδιών, όπου n ο αριθμός που έδωσε στην αρχή ο χρήστης ως μέγιστο αριθμό πληθυσμού. Οι μελωδίες αυτές θα αποτελούνται από όσα μέτρα δήλωσε στην αρχή ο χρήστης και στη συνέχεια θα φυλάγονται σε ένα πίνακα τύπου Phrase στον οποίο θα αποθηκεύονται όλες οι μελωδίες μας. Οι μελωδίες θα δημιουργούνται τυχαία και θα τηρούν τους κανόνες που διατυπώθηκαν προηγουμένως. Θα ακολουθήσει η δημιουργία και των μελωδιών για τη δεύτερη φωνή με πανομοιότυπο τρόπο.

Στη συνέχεια, θα υπολογίζεται η ικανότητα της κάθε μελωδίας σε σχέση με τα κριτήρια που επέλεξε ο χρήστης στην αρχή αθροίζοντας για κάθε κριτήριο πόσο σημαντικό είναι επί το ποσοστό που τηρεί το κριτήριο.

Θα υπολογίζουμε για κάθε μελωδία το ποσοστό ικανότητας και αφού ταξινομήσουμε τις μελωδίες αναλόγως με το συγκεκριμένο ποσοστό, τα αποτελέσματα θα αποθηκεύονται σε ένα πίνακα πραγματικών αριθμών. Για το κριτήριο για τα διαστήματα της οκτάβας, θα ελέγχουμε για κάθε δύο συνεχόμενες νότες στη μελωδία αν το διάστημα τους είναι μια οκτάβα. Αν ναι, τότε θα προσθέτουμε ποσοστό στη συγκεκριμένη μελωδία. Για το κριτήριο αν υπάρχουν συνεχόμενες ίδιες νότες, θα γίνεται έλεγχος σε κάθε μελωδία αν υπάρχει η ίδια νότα για περισσότερες από 3 φορές. Αν ναι, τότε θα αφαιρείται ποσοστό ικανότητας για τη συγκεκριμένη μελωδία. Για το κριτήριο αν υπάρχουν μοτίβα στη μελωδία θα αντιγράφουμε κάθε φορά τη μελωδία μας που θέλουμε να ελέγξουμε σε ένα δεύτερο πίνακα και θα ελέγχουμε την πρώτη φορά το πρώτο μέτρο της πρώτης μελωδίας με όλη τη δεύτερη μελωδία ξεκινώντας όμως από το δεύτερο μέτρο της. Αν βρούμε ότι υπάρχει ίδιο μέτρο στη δεύτερη μελωδία τότε θα γίνει περαιτέρω έλεγχος στη δεύτερη μελωδία αν συμπίπτει το επόμενο μέτρο με το επόμενο μέτρο της πρώτης μελωδίας. Αυτό συνεχίζεται μέχρι να υπάρξουν διαφορετικά μέτρα. Αφού βρούμε μοτίβο στη μελωδία προσθέτουμε ποσοστό στη συνάρτηση ικανότητας. Για το κριτήριο το οποίο ελέγχει αν οι νότες της μελωδίας ανήκουν στην αρχική κλίμακα που επέλεξε ο χρήστης θα ελέγχουμε στον πίνακα που περιέχει τη μελωδία μας αν όλες οι νότες είναι μια από τις νότες της επιλεγμένης κλίμακας. Αν ναι, τότε θα προσθέσουμε ποσοστό

ικανότητας στη συγκεκριμένη μελωδία. Στη συνέχεια θα γίνετε έλεγχος αν η δεύτερη φωνή περιέχει νότες οι οποίες απέχουν από τις νότες της πρώτης μελωδίας διάστημα τρίτης, πέμπτης, ογδόης ή ακόμη αν είναι οι ίδιες νότες. Σε περίπτωση που ισχύει αυτό τότε προσθέτουμε ποσοστό στη μελωδία.

Αφού έχει υπολογισθεί και η συνάρτηση ικανότητας για κάθε μελωδία, θα γίνεται έλεγχος αν ο αλγόριθμος πρέπει να τερματίσει ή να συνεχιστεί. Ο αλγόριθμος τελειώνει αν έχει επαναληφθεί όσες φορές έχει ορίσει στην αρχή ο χρήστης να εκτελεστεί ή αν έχει πάρει το επιθυμητό αποτέλεσμα από τη συνάρτηση ικανότητας σε σύγκριση με τον αριθμό που το έδωσε ο χρήστης. Ακόμη ο αλγόριθμος τερματίζει και όταν το θελήσει ο χρήστης. Σε περίπτωση που θα πρέπει να ελέγξουμε αν πήραμε το επιθυμητό ποσοστό που ζήτησε ο χρήστης, θα υπολογιστεί το ποσοστό ικανότητας για κάθε μελωδία και θα αποθηκευτεί στον πίνακα, στη συνέχεια θα ταξινομούμε τον πίνακα αυτό κατά φθίνουσα σειρά και θα γίνεται έλεγχος αν η πρώτη μελωδία έχει ποσοστό ικανότητας μεγαλύτερο ή ίσο από το ποσοστό που έδωσε στην αρχή ο χρήστης. Αν ναι, τότε τερματίζει ο αλγόριθμος, του επιστρέφεται το αποτέλεσμα και ρωτά το χρήστη αν επιθυμεί να επαναληφθεί ο αλγόριθμος, διαφορετικά αν δεν έχουμε πάρει το επιθυμητό αποτέλεσμα θα συνεχίζεται ο αλγόριθμος για να βρει την επιθυμητή λύση.

Αν ο αλγόριθμος δεν τερματίσει θα επιλέγονται ποιες μελωδίες θα αποτελούν τον πληθυσμό για την επόμενη φορά. Θα ελέγχει στον πίνακα που περιέχει τα ποσοστά ικανότητας ποιες μελωδίες έχουν ποσοστό το οποίο πλησιάζει στο επιθυμητό ποσοστό που έθεσε αρχικά ο χρήστης. Από εκείνες όσες έχουν αυτό το ποσοστό θα επιλέγουμε τις ν μελωδίες και θα τις αποθηκεύουμε σε ένα νέο πίνακα τύπου Phrase για να δημιουργηθεί ο καινούργιος πληθυσμός μέσω του οποίου θα επιλεγούν οι μελωδίες με το μεγαλύτερο ποσοστό ικανότητας για να αποτελέσουν τους γονείς για να δεχτούν την εκτέλεση των γενετικών λειτουργιών της διασταύρωσης ενός σημείου, της διασταύρωσης δύο σημείων και τη γενετική λειτουργία της μεταλλαγής.

Στο τέλος του προγράμματος παρουσιάζεται μια μελωδία στο χρήστη η οποία δίνει το επιθυμητό αποτέλεσμα τόσο ακουστικό όσο και αισθητικό και του δίνεται η δυνατότητα να ακούσει τη μελωδία. Σε περίπτωση που δεν έχει βρεθεί ένα ικανοποιητικό ποσοστό για τη συνάρτηση ικανότητας ή σε περίπτωση που ο αλγόριθμος έχει επαναληφθεί σε μέγιστο βαθμό τότε παρουσιάζεται στο χρήστη μια φόρμα που του αναφέρει ότι δυστυχώς δεν έχει βρεθεί ικανοποιητικό αποτέλεσμα και του παρουσιάζει την καλύτερη μελωδία που βρέθηκε μέχρι τη συγκεκριμένη χρονική στιγμή.

Κεφάλαιο 5

Υλοποίηση

5.1 Υλοποίηση και εξήγηση προγράμματος

Στο κεφάλαιο που ακολουθεί θα γίνει μια περιγραφή των σημαντικών συναρτήσεων που υλοποιήθηκαν στο πρόγραμμα. Ο συνολικός ο κώδικας περιλαμβάνεται στο τέλος της διπλωματικής εργασίας.

5.1.1 Αρχικοποίηση

Ακολουθούν οι συναρτήσεις μέσω των οποίων δημιουργήθηκε ο αρχικός πληθυσμός του γενετικού αλγορίθμου.

Κλάση: Initialise_variables

Συνάρτηση: initialise_population

```
public Vector<Phrase> initialise_population(int populationSize,
                        int num_of_bars, String scale)
```

Η συνάρτηση initialise_population() παίρνει ως είσοδο το μέγεθος του πληθυσμού, τον αριθμό των μέτρων που θα περιέχει κάθε μελωδία και την αρχική κλίμακα που θα δημιουργηθεί η μελωδία. Στη συνέχεια γίνεται κλήση της συνάρτησης compose_phrase για όσες φορές είναι ο αριθμός του πληθυσμού, η οποία δημιουργεί τις τυχαίες μελωδίες για την πρώτη φωνή, για τις συγχορδίες και για τη δεύτερη φωνή και στο τέλος της συνάρτησης επιστρέφεται το vector initial_population το οποίο περιέχει τις μελωδίες του αρχικού πληθυσμού.

Συνάρτηση: second_voice_return()

```
public Vector<Phrase> second_voice_return()
```

Η συνάρτηση second_voice_return() απλώς επιστρέφει τις μελωδίες της δεύτερης φωνής που δημιουργήθηκαν αρχικά στο vector second_voice1.

Κλάση: `Composer`

Συνάρτηση: `return_root`

```
int return_root(String scale)
```

Η `return_root` συνάρτηση παίρνει ως είσοδο την αρχική μας κλίμακα και επιστρέφει τη θεμέλιο (πρώτη) νότα της κλίμακας.

Συνάρτηση: `compose_phrase`

```
public Phrase compose_phrase(int num_of_bars, String scale)
```

Η compose_phrase συνάρτηση παίρνει ως είσοδο τον αριθμό των μέτρων που ζήτησε ο χρήστης να περιέχει η μελωδία και την αρχική κλίμακα. Στη συνέχεια καλείται η συνάρτηση compose_melody για να δημιουργήσει τυχαία μελωδία και στο τέλος επιστρέφεται η συγκεκριμένη μελωδία.

Συνάρτηση: compose_melody

```
public Phrase compose_melody(int num_of_bars, String scale)
```

Αρχικά, δημιουργείται τυχαία μια νότα η οποία μπορεί να πάρει τιμές από NTO χαμηλής φωνής μέχρι και MI σε ψηλής φωνής με τον εξής τρόπο:

```
int pitch = (int) (Math.random() * 17) + 60;
```

Η αξία της κάθε νότας πάλι επιλέγεται τυχαία. Αν η τιμή που λαμβάνουμε είναι μικρότερη από 1.0 και μεγαλύτερη από 0.75 τότε η αξία της νότας θα είναι τέταρτο. Αν η τιμή είναι μικρότερη από 0.75 και μεγαλύτερη από 0.5 τότε η αξία παίρνει την τιμή του ογδόου. Σε περίπτωση που η τυχαία τιμή που θα λάβουμε είναι μεταξύ του 0.5 και 0.25 τότε έχουμε νότα με αξία μισού αλλιώς σε οποιαδήποτε άλλη περίπτωση έχουμε δέκατο-έκτο. Ακολουθεί έλεγχος αν είναι η πρώτη νότα της μελωδίας μας τότε βάζουμε τη θεμέλιο νότα της μελωδίας και η αξία της θα είναι τέταρτο, σε περίπτωση που είναι τα δύο τελευταία μέτρα της μελωδίας οι τέσσερις τελευταίες νότες της μελωδίας θα είναι επιτονική–θεμέλιος-δεσπόζουσα–θεμέλιος νότες της κλίμακας. Συγκεκριμένα ο κώδικας που ακολουθεί υλοποιεί την πτώση της μελωδίας.

```
if (i == num_of_bars - 2) {
                                        n = new Note(notes[1], 2.0);
                                        melodyPhrase.addNote(n);
                                        beatCounter += 2.0;

                                        n1 = new Note(n.getPitch() + 7,
CROTCHET);
                                        second_voice.addNote(n1);
                                        n1 = new Note(n.getPitch(),
CROTCHET);
                                        second_voice.addNote(n1);
                                        beatCounter2 += 2.0;

                                        n = new Note(notes[0], 1.0);
                                        melodyPhrase.addNote(n);
                                        beatCounter += 1.0;
```

```
                                        n1 = new Note(notes[4], 0.5);
                                        second_voice.addNote(n1);
                                        n1 = new Note(notes[2], 0.5);
                                        second_voice.addNote(n1);
                                        beatCounter2 += 1.0;

                                        n = new Note(notes[4], 1.0);
                                        melodyPhrase.addNote(n);
                                        beatCounter += 1.0;

                                        n1 = new Note(n.getPitch() + 7,
   0.5);
                                        second_voice.addNote(n1);
                                        n1 = new Note(n.getPitch(), 0.5);
                                        second_voice.addNote(n1);
                                        beatCounter2 += 1.0;
                                }
```

Στη συνέχεια σε περίπτωση που έχουμε πιθανότητα μικρότερη ή ίση του 0.2 τότε δημιουργούμε μοτίβο αλλιώς σε αντίθετη περίπτωση δημιουργούμε νότα συγκεκριμένης αξίας. Η νότα που θα δημιουργηθεί στην πρώτη θέση θα αποτελεί και την πρώτη νότα της αντίστοιχης συγχορδίας που θα δημιουργηθεί αμέσως μετά. Ταυτόχρονα δημιουργείται με τον ίδιο τρόπο η μελωδία για τη δεύτερη φωνή. Η διαδικασία επαναλαμβάνεται για όλα τα μέτρα και στο τέλος επιστρέφεται η μελωδία για την πρώτη φωνή.

Συνάρτηση: `return_second()`

```
Phrase return_second()
```

Η συνάρτηση return_second επιστρέφει την μελωδία που δημιουργήθηκε για τη δεύτερη φωνή.

5.1.2 Αναπαραγωγή

Στην συνέχεια εξηγούνται οι συναρτήσεις οι οποίες αναπαράγουν καινούργιο πληθυσμό σε κάθε επανάληψη του αλγορίθμου.

Κλάση: `Survivor`

Συνάρτηση: `choose_survivor`

```
Vector<Phrase> choose_survivor(Vector<Phrase> population,
            Vector<Phrase> second, Vector<Double> fitnessArray,
```

```
                                    int population_size)
```

Η συνάρτηση παίρνει ως είσοδο ένα vector με τις μελωδίες της πρώτης φωνής, ένα vector με τις μελωδίες της δεύτερης φωνής, ένα vector με τα ποσοστά ικανότητας της κάθε μελωδίας και το μέγεθος του αρχικού πληθυσμού. Στη συνέχεια ταξινομεί τις μελωδίες κατά φθίνουσα σειρά σε σχέση με τα ποσοστά ικανότητας και μετά επιλέγει τις πρώτες population size μελωδίες για να επιστρέψει στο τέλος το vector με τις νέες μελωδίες για την πρώτη φωνή.

Συνάρτηση: `return_new_population_second`

```
public Vector<Phrase> return_new_population_second()
```

Η συνάρτηση return_new_population_second επιστρέφει τις πρώτες population size μελωδίες για την δεύτερη φωνή.

Συνάρτηση: `return_fitness`

```
Vector<Double> return_fitness(Vector<Phrase> population,
             Vector<Double> fitnessArray, int population_size)
```

Η συνάρτηση παίρνει ως είσοδο ένα vector με τις μελωδίες και ένα vector με τα ποσοστά ικανότητας της κάθε μελωδίας και το μέγεθος του αρχικού πληθυσμού, και αφού ταξινομήσει τα ποσοστά ικανότητας κατά φθίνουσα σειρά, στη συνέχεια επιλέγει τα πρώτα population size ποσοστά ικανότητας και επιστρέφει το vector που τα περιέχει.

5.1.3 Συνάρτηση Ικανότητας

Η συνάρτηση ικανότητας αποτελεί το πιο σημαντικό μέρος του γενετικού αλγορίθμου αφού μέσω αυτής αξιολογείται πόσο κατάλληλη είναι η κάθε υποψήφια λύση για να αποτελέσει την τελική λύση. Στο συγκεκριμένο γενετικό αλγόριθμο η καταλληλότητα του κάθε χρωμοσώματος αξιολογείται μέσω της μαθηματικής πράξης $\sum_{i=1}^{n} w_i x_i$, όπου wi είναι το βάρος που λαμβάνει το ι-οστό κριτήριο αξιολόγησης και xi αποτελεί τον αριθμό των νότων που πληρούν θετικά το συγκεκριμένο κριτήριο που μελετείται κάθε φορά. Παραδείγματος

χάριν, την στιγμή που αξιολογούμε την μελωδία για το κριτήριο της τονικότητας στην μελωδία η συνάρτηση μας επιστρέφει τον αριθμό των νότων που ανήκουν στην αρχική κλίμακα **x = intervals_oktave(pitches)** και στην συνέχεια πολλαπλασιάζουμε αυτό τον αριθμό επί το βάρος που θέσαμε για το συγκεκριμένο κριτήριο **x = x * kr3**. Στο τέλος τα αποτελέσματα που μας δίνουν αυτές οι πράξεις για όλα τα κριτήρια αξιολόγησης προστίθονται για να λάβουμε στο τέλος το τελικό ποσοστό ικανότητας για κάθε λύση του προβλήματος. Παρακάτω ακολουθούν οι συναρτήσεις οι οποίες αξιολογούν την ικανότητα της κάθε μελωδίας.

Κλάση: `Evaluater`

Συνάρτηση: `evaluate_fitness`

```
public static Vector<Double> evaluate_fitness(Vector<Phrase> population,
  Vector<Phrase> population_s, int num_of_bars, String scale,double rythm1,
int kr1, int kr2, int kr3, int kr4, int kr5, int kr6)
```

Η συνάρτηση evaluate_fitness παίρνει ως είσοδο τις μελωδίες για την πρώτη φωνή, τις μελωδίες για τη δεύτερη φωνή, τον αριθμό μέτρων για τις μελωδίες, την αρχική μας κλίμακα, και το ρυθμό των μελωδιών, και για όλα τα κριτήρια στέλνεται αν ο χρήστης θέλει να ληφθούν υπόψη ή αν δεν θέλει να υπολογιστούν. Στην αρχή τοποθετούμε σε ένα πίνακα ακεραίων τις νότες κάθε μελωδίας και σε ένα πίνακα πραγματικών τις αξίες της κάθε νότας. Στη συνέχεια γίνεται κλήση της κάθε συνάρτησης κριτηρίων και κάθε φορά πολλαπλασιάζουμε το ποσοστό ικανότητας για το κάθε κριτήριο επί τον αριθμό που μας επιστρέφει η καθεμία από αυτές τις συναρτήσεις, έτσι ώστε στο τέλος να δημιουργηθεί το τελικό ποσοστό ικανότητας για την κάθε μελωδία. Ο αριθμός που επιστρέφουν οι συναρτήσεις κριτηρίων υπολογίζεται αυξάνοντας ένα μετρητή κάθε φορά που βρίσκει μη κατάλληλη νότα. Στη συνέχεια ο αριθμός που θα προκύψει αφαιρείται από το συνολικό αριθμό νότων που έχει η μελωδία που μελετούμε. Ο καινούργιος αριθμός που θα προκύψει θα αφαιρεθεί από το 100 και θα διαιρεθεί με το συνολικό αριθμό νότων έτσι ώστε να βρεθεί αναλογία. Συγκεκριμένα εκτελείται η παρακάτω πράξη, όπου pitches.length ο συνολικός αριθμός νότων της μελωδίας, num_pos είναι ο αριθμός των νότων που δεν ικανοποιούν το συγκεκριμένο κριτήριο.

```
fitness = (100 * (pitches.length - num_pos)) / pitches.length;
```

5.1.3.1 Κριτήρια αξιολόγησης:

Κριτήριο Τονικότητας:

Στο συγκεκριμένο κριτήριο στην αρχή παίρνουμε τις νότες της αρχικής μας κλίμακας και ελέγχουμε για κάθε μια νότα της μελωδίας μας αν είναι κάποια από αυτές.

```
   for (j = 0; j < scale1.length; j++) {
                           if ((pitches[i] > 0)
                                       && (pitches[i] == scale1[j]
                                             || pitches[i] == scale1[j] + 12
|| pitches[i] == scale1[j] + 24))
```

Αν βρήκαμε νότα η οποία δεν ανήκει στην αρχική κλίμακα τότε αυξάνουμε τον μετρητή num_pos. Στο τέλος υπολογίζουμε πόσες είναι οι 'καλές νότες', δηλαδή πόσες ανήκουν στην αρχική κλίμακα μέσω της πράξης pitches.length - num_pos και μετά βρίσκουμε την αναλογία σε σχέση με 100 νότες όπου και είναι ο μέγιστος αριθμός νότων μιας μελωδίας. Συγκεκριμένα εκτελείται ο εξής κώδικας:

```
fitness = (100 * (pitches.length - num_pos)) / pitches.length;
```

Κριτήριο Ίδιες Συνεχόμενες Νότες:

Στο κριτήριο για ύπαρξη ίδιων συνεχόμενων νότων στην μελωδία ξεκινώντας από την πρώτη νότα της μελωδίας ελέγχουμε τις επόμενες δύο νότες αν είναι οι ίδιες με την πρώτη νότα.

```
   for (j = 0; j < pitches.length - 3; j++) {

                     if (pitches[j] > 0
                                 && (pitches[j] == pitches[j + 1]
                                           && pitches[j + 1] ==
   pitches[j + 2] && pitches[j] > 0)) {
                           num_pos++;
                     }
               }
```

Σε περίπτωση που αυτό συμβαίνει αυξάνεται ο μετρητής `num_pos` και μετά υπολογίζουμε πόσες φορές στην μελωδία μας συμβαίνει να υπάρχουν τρεις συνεχόμενες νότες. Στο τέλος βρίσκουμε την αναλογία σε σχέση με 100 νότες όπου και είναι ο μέγιστος αριθμός νότων μιας μελωδίας. Συγκεκριμένα εκτελείται ο εξής κώδικας:

```
fitness = (100 * (pitches.length - num_pos)) / pitches.length;
```

Κριτήριο Συνεχόμενες Παύσεις:

Στο κριτήριο για έλεγχο αν υπάρχουν συνεχόμενες παύσεις στην μελωδία ελέγχουμε για κάθε φορά που υπάρχει παύση στην μελωδία αν οι επόμενες δύο νότες είναι και αυτές παύσεις μέσω του παρακάτω κώδικα:

```
for (j = 0; j < pitches.length - 3; j++) {

                    if ((pitches[j] < 0 && pitches[j + 1] < 0 &&
pitches[j + 2] < 0)
                                    || (pitches[j] < 0 && pitches[j + 1] <
0 && rythm[j] == 2.0 && (rythm[j + 1] == 2.0 || rythm[j + 1] == 1.0))
                                    || (pitches[j] < 0 && pitches[j + 1] < 0 &&
rythm[j] == 1.0 && (rythm[j + 1] == 2.0)))
```

Σε περίπτωση που υπάρχουν συνεχόμενες παύσεις αυξάνεται ο μετρητής `num_pos` και μετά υπολογίζουμε πόσες φορές στην μελωδία μας συμβαίνει το ίδιο. Ακολουθεί στο τέλος η αναλογία των περιπτώσεων που βρήκαμε σε σχέση με 100 νότες όπου και είναι ο μέγιστος αριθμός νότων μιας μελωδίας μέσω του εξής κώδικα:

```
fitness = (100 * (pitches.length - num_pos)) / pitches.length;
```

Κριτήριο Για Ύπαρξη Μοτίβων:

Στην συγκεκριμένη περίπτωση αξιολόγησης κριτηρίου αρχικά εντοπίζουμε τις θέσεις όπου ξεκινά το κάθε μέτρο και πότε τελειώνει εκτελώντας τον παρακάτω κώδικα.

```
 for (i = 0; i < num_of_bars; i++) {

                    beatCounter = 0.0;

                    beatCounter += rythm[k];
                    k++;

                    while (beatCounter != 4.0) {
                          beatCounter += rythm[k];
                          k++;
                    }
                    metres[i] = k - 1;
              }
```

Στην συνέχεια μετράμε πόσες νότες περιέχονται σε κάθε μέτρο και για κάθε δύο μέτρα τα οποία έχουν ίδιο αριθμό από νότες ελέγχουμε αν οι αντίστοιχες τους νότες απέχουν διάστημα πέμπτης μεταξύ τους. Σε περίπτωση που αυτό συμβαίνει αυξάνεται ο μετρητής για ύπαρξη μοτίβου.

```
for (i = 0; i < num_of_bars - 2; i++) {
                    if (metres2[i] == metres2[num_of_bars - 2]) {// an
exoun to idio
                          // mikos ta duo
```

```
// metra
count = metres3[num_of_bars - 2];
counter = 0;

for (n = metres3[i]; n <= metres[i]; n++) {
      if (pitches[n] == pitches[count] + 7)
            counter++;
      else
            break;
      count++;
}
```

Κριτήριο Για Επιτρεπτά Και Μη Επιτρεπτά Διαστήματα:

Ο έλεγχος για την ύπαρξη διαστημάτων οκτάβας στην μελωδία και διαστημάτων πέραν της οκτάβας γίνεται ταυτόχρονα. Συγκεκριμένα βρίσκουμε την απόσταση μεταξύ κάθε δύο συνεχόμενων νότων μέσω της ακόλουθης πράξης result = pitches[j] - pitches[j + 1]. Στην συνέχεια το αποτέλεσμα που θα λάβουμε ελέγχουμε αν ισούται με δώδεκα, το οποίο ο συγκεκριμένος αριθμός αναπαριστά το διάστημα οκτάβας. Αν ισούται, τότε αυξάνουμε τον μετρητή num_pos2, αλλιώς ελέγχουμε αν το αποτέλεσμα είναι πιο μεγάλο από δώδεκα που σημαίνει ότι οι δύο νότες απέχουν απόσταση μεγαλύτερη της οκτάβας. Σε τέτοια περίπτωση αυξάνουμε τον μετρητή num_pos. Στο τέλος υπολογίζουμε την αναλογία σε εκατό νότες σε σχέση με τον μετρητή που περιέχει τον αριθμό των διαστημάτων οκτάβας, και την αναλογία σε σχέση με τον μετρητή που περιέχει τον αριθμό των διαστημάτων πέραν της οκτάβας.

Κριτήριο Συνέπειας Μεταξύ Πρώτης Και Δεύτερης Φωνής:

Στο συγκεκριμένο κριτήριο αρχικά ελέγχουμε αν οι αντίστοιχες νότες μεταξύ πρώτης και δεύτερης φωνής είναι οι ίδιες ή απέχουν διάστημα οκτάβας. Σε περίπτωση που συμβαίνει κάτι τέτοιο αυξάνουμε τον μετρητή num_pos.

```
if (pitches_first[i] > 0
                        && pitches_second[i] > 0
                        && (pitches_first[i] ==
pitches_second[i]
                                    || pitches_first[i] ==
pitches_second[i] - 12 || pitches_first[i] - 12 ==
pitches_second[i])) {
                  num_pos++;
            }
```

Μετά ελέγχουμε αν οι αντίστοιχες νότες απέχουν διάστημα πέμπτης ή τρίτης.

```
if (pitches_first[i] > 0
                                && pitches_second[i] > 0
                                && ((pitches_first[i] ==
pitches_second[i] - 7) || (pitches_first[i] - 7 ==
pitches_second[i]))) {
                          num_pos++;
            }
```

Στο τέλος υπολογίζουμε και πάλι την αναλογία των 'καλών νότων' δηλαδή αυτών που απέχουν μεταξύ τους διαστήματα τρίτης, πέμπτης ή όγδοης σε σχέση με το εκατό όπου είναι και ο μέγιστος αριθμός νότων που μπορεί να υπάρξουν σε μια μελωδία.

5.1.4 Γενετικές Λειτουργίες

Σε αυτό το σημείο περιγράφονται οι γενετικές λειτουργίες που χρησιμοποιήθηκαν στον γενετικό αλγόριθμο.

5.1.4.1 Μετάλλαξη

Ως πρώτη γενετική λειτουργία που αναπτύχθηκε είναι αυτή της μετάλλαξης.

Κλάση: `Mutate`

Συνάρτηση: `mutate_function`

```
Phrase mutate_function(Phrase table, Phrase second, String scale)
```

Η συνάρτηση mutate_function παίρνει ως είσοδο τη συγκεκριμένη μελωδία τόσο για την πρώτη φωνή όσο και για την δεύτερη, οι οποίες θα δεχτούν τη γενετική λειτουργία της μεταλλαγής και τοποθετεί σε ένα πίνακα τις νότες της και τις αξίες των νότων για κάθε μελωδία. Στη συνέχεια βρίσκουμε μια τυχαία νότα για να γίνει η αλλαγή. Αν είναι η πρώτη νότα της μελωδίας δεν γίνεται αλλαγή γιατί θέλουμε να παραμείνει η θεμέλιος νότα της κλίμακας, αν είναι η τελική πτώση της μελωδίας πάλι δεν κάνουμε αλλαγή, αλλιώς ελέγχουμε αν η συγκεκριμένη νότα ανήκει στην αρχική κλίμακα. Σε περίπτωση που συμβαίνει κάτι τέτοιο

τότε επιλέγουμε άλλη νότα της μελωδίας για να αλλάξει τιμή. Ο συγκεκριμένος έλεγχος που ελέγχει τη συγκεκριμένη νότα που επιλέγηκε για μετάλλαξη ακολουθεί παρακάτω.

```
for (int j = 0; j < scale1.length && flag == 0; j++) {
                                if (pitches_s[i] == scale1[j]
                                            || pitches_s[i] ==
scale1[j] + 12)// i
                                // nota
                                // anikei
                                // stin
                                // klimaka
                                {
                                      flag = 1;
                                }
                          }
```

Αν δεν βρεθεί νότα στη μελωδία της πρώτης φωνής η διαδικασία εντοπισμού νότας για μεταλλαγή γίνεται στη δεύτερη φωνή. Όταν θα γίνει η μεταλλαγή ελέγχουμε αν η προηγούμενη νότα από αυτή που θα γίνει μεταλλαγή ανήκει στην αρχική κλίμακα. Αν αυτό συμβαίνει τότε η καινούργια μας νότα θα είναι η επόμενη νότα της κλίμακας σε σχέση με την προηγούμενη. Δηλαδή αν η προηγούμενη νότα από αυτή που θα γίνει μετάλλαξη ήταν η ΝΤΟ, τότε η καινούργια νότα που θα προκύψει θα είναι η ΡΕ. Σε περίπτωση που η προηγούμενη νότα δεν ανήκει στην αρχική κλίμακα, τότε γίνεται ο ίδιος έλεγχος με την επόμενη νότα από αυτή που θα αλλάξει, με τη διαφορά ότι αν η επόμενη νότα ανήκει στην αρχική κλίμακα τότε η καινούργια νότα θα είναι η προηγούμενη της κλίμακας. Δηλαδή αν η επόμενη νότα σε σχέση με αυτή που θα γίνει μεταλλαγή είναι το ΡΕ τότε η καινούργια νότα που θα προκύψει είναι η ΝΤΟ. Στο τέλος της συνάρτησης επιστρέφεται η μελωδία της πρώτης που θα προκύψει μετά τη γενετική λειτουργία της μετάλλαξης.

Συνάρτηση: `return_offspring2`

```
Phrase return_offspring2()
```

Η συνάρτηση return_offspring2 επιστρέφει τη μελωδία της δεύτερης φωνής που θα προκύψει μετά τη γενετική λειτουργία της μετάλλαξης.

5.1.4.2 Διασταύρωση

Ως δεύτερος γενετικός τελεστής που αναπτύχθηκε είναι η διασταύρωση ενός και δύο σημείων.

Κλάση: `Crossover`

Συνάρτηση: `get_point`

```
private static int get_point(int num_of_bars)
```

Η συνάρτηση `get_point` παίρνει ως είσοδο τον αριθμό μέτρων της μελωδίας και επιστρέφει ένα αριθμό ο οποίος αντιπροσωπεύει ένα τυχαίο μέτρο.

Συνάρτηση: `execute_crossover`

```
static Phrase[] execute_crossover(Phrase parent1, Phrase parent2,
  int num_of_bars, int rythm1, String scale, int flag)
```

Η συγκεκριμένη συνάρτηση παίρνει ως είσοδο τους δύο γονείς, τον αριθμό των μέτρων των μελωδίων, το ρυθμό της μελωδίας, και την αρχική κλίμακα. Και στη συνέχεια καλεί τη συνάρτηση one_point_crossover η οποία εκτελεί την ενός σημείου διασταύρωση. Στο τέλος επιστρέφονται οι γονείς με τις αλλαγές τους.

Συνάρτηση: `one_point_crossover`

```
static Phrase[] one_point_crossover(Phrase parent1, Phrase parent2,
  int num_of_bars, int rythm1, String scale, int flag)
```

Η συνάρτηση one_point_crossover παίρνει ως είσοδο τους δύο γονείς, τον αριθμό μέτρων, το ρυθμό της μελωδίας και την αρχική κλίμακα. Αντιγράφουμε τις νότες του κάθε γονέα σε πίνακες ακεραίων και τις αξίες τους σε ένα πίνακα πραγματικών αριθμών. Μετά παίρνουμε ένα τυχαίο σημείο για να γίνει η διασταύρωση. Από την αρχή των μελωδιών μέχρι το συγκεκριμένο σημείων αντιγράφουμε τις μελωδίες σε Phrase. Στη συνέχεια αντιγράφουμε την υπόλοιπη πρώτη μελωδία στο δεύτερο γονέα και την υπόλοιπη δεύτερη μελωδία στον πρώτο γονέα. Η αντιγραφή του δεύτερου γονέα στον πρώτο γονέα παρουσιάζεται στη συνέχεια.

```
for (i = k2; i < pitches2.length; i++) {
```

```
                    Note n = new Note(pitches2[i], value2[i]);
                    n.setRhythmValue(value2[i]);
                    if (klimaka == 1) {
                          if (pitches2[i] < 0)
                                pitches2[i] = previous2;

                          previous2 = pitches2[i];
                    } else {
                          if (pitches2[i] < 0)
                                pitches2[i] = previous2;

                          previous2 = pitches2[i];
                    }

                    new_p1.addNote(n);
        }
```

Στο τέλος της συνάρτησης επιστρέφουμε τους δύο γονείς.

Συνάρτηση: `two_point_crossover`

```
static Phrase[] two_point_crossover(Phrase parent1, Phrase parent2,
              int num_of_bars, int rythm1, int rythm2, String scale)
```

Η συνάρτηση `two_point_crossover` παίρνει ως είσοδο τους δύο γονείς, τον αριθμό μέτρων, το ρυθμό της μελωδίας και την αρχική κλίμακα. Αντιγράφουμε τις νότες του κάθε γονέα σε πίνακες ακεραίων και τις αξίες τους σε ένα πίνακα πραγματικών αριθμών. Μετά παίρνουμε δύο τυχαία σημεία για να γίνει η διασταύρωση. Από την αρχή των μελωδιών μέχρι το πρώτο σημείο αντιγράφουμε τις μελωδίες σε Phrase. Στη συνέχεια αντιγράφουμε την υπόλοιπη πρώτη μελωδία μέχρι το δεύτερο σημείο στο δεύτερο γονέα και την υπόλοιπη δεύτερη μελωδία μέχρι το δεύτερο σημείο στον πρώτο γονέα. Η υπόλοιπη μελωδία από το δεύτερο σημείο μέχρι το τέλος του πρώτου γονέα μένει η ίδια όπου αντίστοιχα συμβαίνει το ίδιο και με το δεύτερο γονέα. Στο τέλος επιστρέφουμε τους δύο γονείς.

5.1.5 Πλατφόρμα Διεπαφής

Συνεχίζει μια περιγράφεται του τρόπου με τον οποίο δημιουργήθηκε η πλατφόρμα διεπαφής η οποία αλληλεπιδρά με τον χρήστη.

Κλάση: `genetic_algorithm`

Συνάρτηση: `genetic_algorithm`

```
public genetic_algorithm()
```

Στη συνάρτηση `genetic_algorithm` δημιουργείται όλο το interface του προγράμματος. Στη φόρμα που παρουσιάζεται στο χρήστη του δίνεται η δυνατότητα να επιλέξει τον αριθμό του αρχικού πληθυσμού, το μέγιστο αριθμό επαναλήψεων του αλγορίθμου, το μέγιστο ποσοστό ικανότητας που θέλει να επιτύχει ο αλγόριθμος, τον αριθμό των μέτρων που θέλει να έχει η κάθε μελωδία, ποια κριτήρια θέλει να ληφθούν υπόψη στην αξιολόγηση των μελωδιών, την αρχική κλίμακα των μελωδιών και ποιο όργανο θέλει να του εκτελέσει την τελική μελωδία που θα του επιστρέψει ο αλγόριθμος. Συγκεκριμένα παρουσιάζεται στο χρήστη η ακόλουθη φόρμα μέσω της οποίας κάνει τις επιλογές του.

Εικόνα 16: Η φόρμα που εμφανίζεται στο χρήστη στην αρχή του αλγορίθμου.

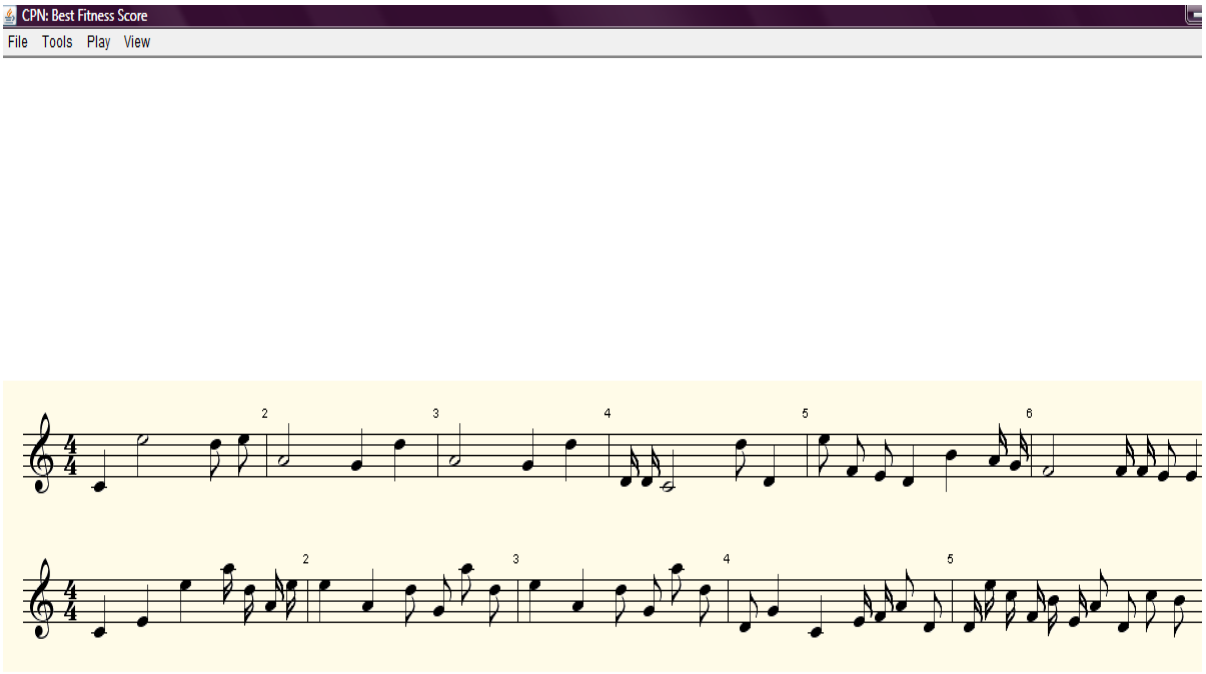
5.1.6 Γενετικός αλγόριθμος

Μετά ακολουθεί μια εκτενής περιγραφή της υλοποίησης του γενετικού αλγορίθμου.

Συνάρτηση: `run_algorithm`

```
static void run_algorithm(int population_size, int num_of_bars,
  String scale, int loop, double percentage, int kr1, int kr2,
              int kr3, int kr4, int kr5, int kr6)
```

Αρχικά γίνεται κλήση της συνάρτησης initialise_population η οποία αρχικοποιεί τον πληθυσμό μας, μετά αξιολογεί τις μελωδίες που δημιουργήθηκαν μέσω της συνάρτησης *evaluate_fitness,* και ακολουθεί η κλήση της συνάρτησης terminator_algorithm η οποία αποφασίζει αν θα τερματίσει ο αλγόριθμος, αν επιστρέψει true τότε ο αλγόριθμος τερματίζει και παρουσιάζονται τα αποτελέσματα. Αλλιώς, εφόσον δεν επιστρέφει true η συγκεκριμένη συνάρτηση, αυξάνεται ο δείκτης των επαναλήψεων, και αν είμαστε στη πρώτη επανάληψη επιλέγουμε επιζώντες μέσω της συνάρτησης choose_survivor και αξιολογούμε τις νέες μελωδίες μέσω της συνάρτησης *evaluate_fitness, τα* ποσοστά που επιστρέφει η συγκεκριμένη συνάρτηση τα προσθέτουμε με τα προηγούμενα ποσοστά, και επιλέγουμε τους δύο γονείς με τα μεγαλύτερα ποσοστά ικανότητας. Στη συνέχεια, τυχαία εκτελείται η διαδικασία της μεταλλαγής ή διασταύρωσης. Οι δύο μελωδίες που επιστρέφονται προστίθονται στον πληθυσμό και γίνεται αξιολόγηση του πληθυσμού. Μετά καλείται η συνάρτηση terminator_algorithm όπου αν επιστρέψει true τερματίζουμε, αλλιώς επαναλαμβάνεται ο αλγόριθμος. Στο τέλος του αλγορίθμου εμφανίζεται στο χρήστη η μελωδία που επιλέγηκε ως καλύτερη. Συγκεκριμένα παρουσιάζεται η επόμενη φόρμα.



Εικόνα 17: Παράδειγμα μελωδίας που επιστρέφει ο γενετικός αλγόριθμος.

5.1.7 Τερματισμός

Οι συναρτήσεις της μέσω των οποίων ο γενετικός αλγόριθμος τερματίζει σχολιάζονται στην συνέχεια.

Κλάση: Terminator

Συνάρτηση: execute_iteration

```
static boolean execute_iteration(int max_iteration, int iteration)
```

Η συνάρτηση execute_iteration παίρνει ως είσοδο τον αριθμό που επαναλήφθηκε ο αλγόριθμος και το μέγιστο αριθμό που ζήτησε ο χρήστης να επαναληφθεί ο αλγόριθμος και επιστρέφει true αν ο αλγόριθμος έχει επαναληφθεί τις μέγιστες φορές που ζήτησε ο χρήστης αλλιώς επιστρέφει false. Συγκεκριμένα γίνεται ο παρακάτω έλεγχος:

```
if (iteration == max_iteration)
                    return true;
 return false;
```

Συνάρτηση: `execute_percent`

```
static boolean execute_percent(Vector<Double> fitnessArray,
              double percentage)
```

Η συγκεκριμένη συνάρτηση παίρνει ως είσοδο το vector με τα ποσοστά ικανότητας που έχει η κάθε μελωδία και το μέγιστο ποσοστό ικανότητας που επιθυμεί να έχει η τελική μελωδία ο χρήστης. Στη συνέχεια ελέγχει για κάθε μελωδία αν έχει επιτύχει το συγκεκριμένο ποσοστό ικανότητας ή ακόμη μεγαλύτερο. Στην πρώτη μελωδία που θα εντοπίσει με τη συγκεκριμένη ιδιότητα η συνάρτηση επιστρέφει true. Αν ελέγξει όλες τις μελωδίες και δεν προκύψει επιθυμητό αποτέλεσμα η συνάρτηση επιστρέφει false. Συγκεκριμένα εκτελείται ο ακολούθως κώδικας:

```
for (int i = 0; i < fitnessArray.size(); i++)
                    if (percentage <= fitnessArray.get(i))
                                return true;
```

Συνάρτηση: `solution`

```
static int solution(Vector<Double> fitnessArray, double percentage)
```

Η συνάρτηση `solution` παίρνει ως είσοδο ένα vector με τα ποσοστά ικανότητας και το μέγιστο ποσοστό ικανότητας που επιθυμεί ο χρήστης να επιτευχθεί στη μελωδία και επιστρέφει τη θέση της μελωδίας η οποία έχει επιτύχει το συγκεκριμένο ποσοστό ή ακόμα μεγαλύτερο.

Συνάρτηση: `solution2`

```
static int solution_no2(Vector<Double> fitnessArray)
```

Η συνάρτηση `solution_no2` παίρνει το vector με τα ποσοστά ικανότητας της κάθε μελωδίας και επιστρέφει την θέση της μελωδίας με το καλύτερο ποσοστό μέχρι κάποια συγκεκριμένη στιγμή ταξινομώντας κατά φθίνουσα σειρά τα ποσοστά ικανότητας.

Συνάρτηση: `terminator_algorithm`

```
boolean terminator_algorithm(int max_iteration, double percentage,
            Vector<Double> fitnessArray, int iteration)
```

Η συνάρτηση terminator_algorithm παίρνει ως είσοδο το μέγιστο αριθμό επαναλήψεων που επιθυμεί ο χρήστης, το μέγιστο ποσοστό ικανότητας που επιθυμεί να έχει η τελική μελωδία, το vector με τα ποσοστά ικανότητας των μελωδιών και τον αριθμό που έχει επαναληφθεί ο αλγόριθμος μέχρι στιγμής. Καλώντας τις συναρτήσεις execute_iteration και execute_percent επιστρέφει true αν η μελωδία τερμάτισε επειδή έχει επαναληφθεί τις μέγιστες φορές ή αν επειδή έχει επιτευχθεί το επιθυμητό ποσοστό ικανότητας σε κάποια μελωδία ανάλογα με τις τιμές που επιστρέφουν οι δύο συναρτήσεις.

Κεφάλαιο 6

Συμπεράσματα

6.1 Αποτελέσματα - Συμπεράσματα
6.2 Μελλοντική Εργασία

6.1 Αποτελέσματα

Έχουμε παρατηρήσει ότι στην υλοποίηση του γενετικού αλγορίθμου όσο αφορά την αλγοριθμική σύνθεση το σημαντικότερο μέρος του είναι η συνάρτηση ικανότητας αφού εφαρμόζεται σε κάθε νέα μελωδία που παράγεται έτσι ώστε να μπορέσουμε να εντοπίσουμε αν υπάρχει η επιθυμητή μελωδία η οποία ικανοποιεί το ποσοστό ικανότητας που ζήτησε ο χρήστης στην αρχή του αλγορίθμου και μ' αυτό τον τρόπο να τερματίσει και ο γενετικός αλγόριθμος. Εφαρμόζοντας την συγκεκριμένη συνάρτηση αξιολογούμε την κάθε μελωδία με βάση συγκεκριμένα κριτήρια που εισάγαμε επιστρέφοντας έτσι μεγαλύτερο ποσοστό ικανότητας στην μελωδία η οποία περιέχει σε μεγαλύτερο βαθμό τα κριτήρια που επιθυμούμε και παρουσιάζοντας στο τέλος στον χρήστη της καλύτερη μελωδία. Ακόμη παρατηρούμε ότι αν ο χρήστης επιλέξει να ληφθούν σε μεγαλύτερο βαθμό υπόψη κριτήρια τα οποία αφορούν αποκλειστικά στοιχεία της εποχής Μπαρόκ όπως είναι η ύπαρξη διαστημάτων οκτάβας, η μη ύπαρξη διαστημάτων πέραν της οκτάβας και η ύπαρξη μοτίβων στη μελωδία μας, τότε η μελωδία κληρονομεί αρκετά στοιχεία από την εποχή Μπαρόκ. Όσο αφορά τα κριτήρια για το αν οι νότες ανήκουν στην αρχική μελωδία που επέλεξε ο χρήστης και αν υπάρχουν ίδιες συνεχόμενες νότες δημιουργούν μια μελωδία η οποία ικανοποιεί βασικούς κανόνες των μελωδιών κάθε εποχής όπως και της Μπαρόκ. Όσο έχει να κάνει με τη δεύτερη φωνή παρατηρούμε ότι οι περισσότερες νότες που παράγονται στην τελική μελωδία της δεύτερης φωνής είναι είτε οι ίδιες με την πρώτη φωνή, είτε απέχουν διάστημα τρίτης ή διάστημα πέμπτης από τη δεύτερη φωνή. Επομένως παρατηρούμε ότι αν ο χρήστης επιθυμεί να λάβει μια μελωδία η οποία θα έχει χαρακτηριστικά στοιχεία της εποχής Μπαρόκ αλλά ταυτόχρονα

είναι εξίσου ικανοποιητική σε μουσικά στοιχεία θα επιλέξει όλα τα κριτήρια και η μελωδία που θα προκύψει θα ικανοποιεί σε μεγάλο βαθμό τις ανάγκες του χρήστη.

Στην αρχή της υλοποίησης του αλγορίθμου οι μελωδίες που λαμβάναμε δεν ήταν οι καλύτερες όσο αφορά την αρμονία της μουσικής. Αυτό συνέβαινε εξαιτίας του γεγονότος ότι δεν γινόταν καθόλου έλεγχος μεταξύ της πρώτης και της δεύτερης φωνής. Μετά την εισαγωγή του κριτηρίου αν οι νότες της πρώτης και δεύτερης φωνής είναι οι ίδιες μεταξύ τους ή απέχουν διάστημα τρίτης ή πέμπτης, παρατηρήσαμε ότι τα αποτελέσματα που παίρναμε ήταν περισσότερο ικανοποιητικά. Ακόμη στην αρχή οι μελωδίες που δημιουργούνταν τις περισσότερες φορές περιείχαν νότες οι οποίες δεν ανήκαν στην αρχική κλίμακα. Ο παραπάνω έλεγχος συνδυασμένος με τον έλεγχο για το αν οι νότες της μελωδίας ανήκουν στη συγκεκριμένη κλίμακα είχαν διορθώσει σε αρκετό βαθμό το πρόβλημα επιστρέφοντας μελωδίες οι οποίες τηρούν το συγκεκριμένο κριτήριο.

Ακόμη παρατηρούμε ότι τα καλύτερα αποτελέσματα λαμβάνονται όταν ο χρήστης ζητήσει να επαναληφθεί ο γενετικός αλγόριθμος αρκετές φορές. Με αυτό τον τρόπο η μελωδίες δέχονται συνεχώς βελτιώσεις μέσω των γενετικών λειτουργιών και έτσι κατά την εκτέλεση της συνάρτησης αξιολόγησης λαμβάνονται καλύτερα ποσοστά ικανότητας από τις μελωδίες. Σε περίπτωση που ο χρήστης επιθυμεί η μελωδία να επιτύχει μικρό αριθμό ικανότητας η μελωδία που θα του επιστραφεί δεν θα είναι η καλύτερη σε θέμα αρμονίας αλλά ούτε θα υιοθετεί αρκετά στοιχεία της εποχής Μπαρόκ εξαιτίας του γεγονότος ότι ο αλγόριθμος δεν θα προλάβει να δεχθεί αρκετές επαναλήψεις αφού θα εντοπιστεί σύντομα μελωδία με μικρό ποσοστό ικανότητας. Το ίδιο θα συμβεί σε περίπτωση που ο χρήστης επιθυμεί ο αλγόριθμος να μην επαναληφθεί αρκετές φορές. Αφού δεν θα εκτελεστούν αρκετές γενετικές λειτουργίες στις μελωδίες έτσι ώστε να τις καλυτερεύσουν, η τελική μελωδία που θα επιστραφεί δεν θα είναι αρκετά επιθυμητή όσο αφορά την αρμονία.

6.2 Μελλοντική Εργασία

Η υλοποίηση της συγκεκριμένης διπλωματικής εργασίας χρησιμοποιώντας γενετικούς αλγορίθμους δίνει τη δυνατότητα στον καθένα να αντιληφθεί ότι η δημιουργία μελωδιών οι οποίες κληρονομούν στοιχεία συγκεκριμένης μουσικής εποχής κάνοντας μόνο χρήση ηλεκτρονικών υπολογιστών δεν είναι δύσκολο επίτευγμα. Λόγω του γεγονότος ότι ο συγκεκριμένος γενετικός αλγόριθμος μπορεί εύκολα να επεκταθεί οποιοσδήποτε ο οποίος ασχολείται με υπολογιστές και μουσική έχει τη δυνατότητα για μελλοντική εργασία στη συγκεκριμένη αλγοριθμική σύνθεση χρησιμοποιώντας γενετικούς αλγορίθμους. Συγκεκριμένα

το πρόγραμμα μπορεί να επεκταθεί δίνοντας επιπλέον ευχέρεια στον χρήστη όσο αφορά τα μοτίβα. Ο χρήστης μπορεί να έχει ο ίδιος της δυνατότητα να καθορίσει ποιες θα είναι οι αξίες των νότων που επιθυμεί να αποτελούν το μοτίβο δίνοντας έτσι την ευκαιρία στον χρήστη στο τέλος να λάβει μελωδία η οποία θα έχει και ο ίδιος μια συμβολή στην δημιουργία της. Ακόμη μια επέκταση η οποία θα μπορούσε να γίνει μελλοντικά στο συγκεκριμένο πρόγραμμα είναι όσο αφορά την αισθητική της μελωδίας προσθέτοντας νότες καλλωπισμού, όπως είναι οι τρίλιες, είτε δίνοντας στον χρήστη την δυνατότητα να επιλέξει διαφορετικό μουσικό όργανο για να εκτελεστεί η δεύτερη φωνή. Επιπλέον ο γενετικός αλγόριθμος θα μπορούσε να τρέχει και για την δημιουργία συγχορδιών χωρίς να δημιουργούνται οι συγχορδίες σε σχέση με την αντίστοιχη νότα που υπάρχει στην πρώτη φωνή αλλά δημιουργώντας τις τυχαία στην αρχή και μετά αξιολογώντας τις σε σχέση με τις αντίστοιχες νότες της πρώτης και δεύτερης φωνής.

Βιβλιογραφία:

[1] D. Aschauer, "Algorithmic Composition", Art@Science 2008.

[2] A. Moroni, J. Manzolli and F. V. Zuben, "Composing with interactive genetic algorithms", Brazilian Symposium of Computer Music, 1999.

[3] A. Moroni, J. Manzolli and F. V. Zuben, "Vox Populi: An Interactive Evolutionary System for Algorithmic Music Composition", Leonardo Music Journal vol. 10, pp. 49-54, 2000.

[4] H. Järveläinen, "Algorithmic musical composition", Art@Science, 2000.

[5] J. A. Biles, "GenJam: Evolutionary Computation Gets a Gig", Proceedings of the 2002 Conference for Information Technology Curriculum, 2002.

[6] M. Edwards, "Algorithmic Composition: Computational Thinking in Music DRAFT", http://ddm.caad.ed.ac.uk/staff/michael/algorithmic-composition.pdf .

[7] J. Bewley," Lejaren A. Hiller: Computer Music Pioneer", (2004) http://library.buffalo.edu/music/exhibits/hillerexhibitsummary.pdf .

[8] G. Nierhaus, "Algorithmic Composition. Paradigms of Automated Music Generation", Springer Wien New York, 2009.

[9] Ι. Δ. Χριστοφίλου, "Θεωρία της Μουσικής", Τάξη Πρώτη, Music Lovers, 1985.

[10] Α. Αμαραντίδης, "Μορφολογία της Μουσικής", Μουσικός εκδοτικός οίκος Κ. Παπαγρηγορίου –Χ. Νάκας, 1990.

[11] C. Headington, "Ιστορία της Δυτικής Μουσικής, Από την αρχαιότητα ως τις μέρες μας", Πρώτος Τόμος, Εκδόσεις Gutenberg, 1993.

[12] A. Shadduck, "Music of the Baroque", 2009, http://austinshadduck.com/documents/baroque.pdf .

[13] M. Pelikan, "Genetic Algorithms", MEDAL Report No. 20010007 Encyclopedia of Operations Research and Management Science (EORMS) (Wiley).

[14] Ι. Γ. Τσούλος, "Γενετικοί αλγόριθμοι", Σημειώσεις μαθήματος, http://users.cs.uoi.gr/~itsoulos/genetic.php

[15] Σπυρίδων Δ. Λυκοθανασης, "Γενετικοί Αλγόριθμοι και Εφαρμογές", Τεχνική Νοημοσύνη και εφαρμογές, Τόμος Γ, 2001.

[16] Ε. Φ. Γεωργόπουλος, Σ. Δ. Λυκοθανασης, "Εισαγωγή Στους Γενετικούς αλγορίθμους", Πάτρα, 1999, http://edu.eap.gr/pli/pli31/docs/GAs_introduction.pdf.

[17] G.Wiggins, G. Papadopoulos, S. Phon-Amnuaisuk and A. Tuson, "Evolutionary Methods for Musical Composition", http://www.dai.ed.ac.uk/groups/aimusic/.

Παράρτημα Α

```
/*
 * Onoma: Tsokkou Panayiota
 * A.T: 955953
 * Onoma Arxeiou: genetic_algorithm
 */

//Libraries
1 import javax.swing.JApplet;
import jm.JMC;
import jm.music.data.*;
import jm.midi.*;
import jm.music.tools.*;
import jm.util.*;
import jm.gui.show.ShowScore;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.Dialog;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import jm.midi.event.TimeSig;
import java.util.ArrayList;

public class genetic_algorithm extends Frame implements
ActionListener,
            WindowListener, JMC {

      static private Button keysArray[];
      static private Panel keyPad;
      static private Button keysOrgan[];
      static private Panel keyOrgan;
      static final int barSize = 4;
      static int violin = 0, viola = 0, oboe = 0, flute = 0, piano =
0,
                  tsello = 0;
      String scale = null;
      Frame fr;
      Button bt;
      TextField populationSize;
      TextField numIterations;
      TextField fitnessVal;
      TextField metersVal;
      TextField contentPanel21;
      TextField contentPanel31;
      TextField contentPanel41;
      TextField contentPanel51;
      TextField contentPanel61;
      TextField contentPanel71;
      FileDialog fd;
      static Score s = new Score("Best Fitness Score");

Button runBtn, showBtn;

      // --------------
      // simple main method called when the class in run
      // --------------
```

```
      Event ID;
      public static Vector<Phrase> population_4 = new
Vector<Phrase>();
      public static Vector<Phrase> population_4_s = new
Vector<Phrase>();
      public static Vector<Double> fitness_array_4 = new
Vector<Double>();

      // Constructor
      public genetic_algorithm() {

            // give the window a name
            super("Genetic Algorithm Implementation");
            this.setBackground(new java.awt.Color(241, 227, 253));
            // register the closebox event
            this.addWindowListener(this);
            this.setLocation(3, 3);

            Panel contentPanel = new Panel();
            Panel contentPanel12 = new Panel();
            JLabel jLabel2 = new javax.swing.JLabel();
            jLabel2.setIcon(new javax.swing.ImageIcon(
                        "/C:/Users/User/Desktop/Presentation1.jpg"));
            jLabel2.setBounds(P, P, 2, 2);
            contentPanel12.add(jLabel2, BorderLayout.PAGE_START);
            this.setResizable(false);

            // add the components
            Label popSizeL = new Label("Please Enter Population Size
(1 to 200): ");
            popSizeL.setForeground(new java.awt.Color(102, 0, 102));
            popSizeL.setFont(new java.awt.Font(null, 1, 12));

            Label iterationsNumL = new Label(
                        "Please Enter max number of iterations (1 to
200): ");
            iterationsNumL.setForeground(new java.awt.Color(102, 0,
102));
            iterationsNumL.setFont(new java.awt.Font(null, 1, 12));

            Label fitnessL = new Label("Please Enter max Fitness
Value (>0) : ");
            fitnessL.setForeground(new java.awt.Color(102, 0, 102));
            fitnessL.setFont(new java.awt.Font(null, 1, 12));

            Label meters = new Label("Please Enter meters num (>0) :
");
            meters.setForeground(new java.awt.Color(102, 0, 102));
            meters.setFont(new java.awt.Font(null, 1, 12));

            Panel labelPanel = new Panel();
            labelPanel.setLayout(new GridLayout(0, 1));

            labelPanel.add(popSizeL);
            labelPanel.add(iterationsNumL);
            labelPanel.add(fitnessL);
            labelPanel.add(meters);

            populationSize = new TextField("100");
```

```
            populationSize.setBackground(new java.awt.Color(102, 0,
102));
            populationSize.setForeground(Color.white);
            numIterations = new TextField("100");
            numIterations.setBackground(new java.awt.Color(102, 0,
102));
            numIterations.setForeground(Color.white);
            fitnessVal = new TextField("700");
            fitnessVal.setBackground(new java.awt.Color(102, 0,
102));
            fitnessVal.setForeground(Color.white);
            metersVal = new TextField("5");
            metersVal.setBackground(new java.awt.Color(102, 0, 102));
            metersVal.setForeground(Color.white);

            Panel fieldPanel = new Panel();
            fieldPanel.setLayout(new GridLayout(0, 1));

            fieldPanel.add(populationSize);
            fieldPanel.add(numIterations);
            fieldPanel.add(fitnessVal);
            fieldPanel.add(metersVal);

            Panel contentPane13 = new Panel();
            Panel contentPanel9 = new Panel();
            Panel contentPanel10 = new Panel();

            keyPad = new Panel();
            keyPad.setLayout(new GridLayout(0, 3));
            keysArray = new Button[24];

            Panel labelPanel1 = new Panel();
            labelPanel1.setLayout(new GridLayout(0, 1));
            Label scale = new Label("  Please Enter scale:");
            scale.setForeground(new java.awt.Color(102, 0, 102));
            scale.setFont(new java.awt.Font(null, 1, 12));
            labelPanel1.add(scale);

            // dimiourgoume ta koumpia

            keysArray[0] = new Button("Cmajor");
            keysArray[1] = new Button("Cminor");
            keysArray[2] = new Button("C#major");
            keysArray[3] = new Button("C#minor");

            keysArray[4] = new Button("Dmajor");
            keysArray[5] = new Button("Dminor");
            keysArray[6] = new Button("D#major");
            keysArray[7] = new Button("D#minor");

            keysArray[8] = new Button("Emajor");
            keysArray[9] = new Button("Eminor");

            keysArray[10] = new Button("Fmajor");
            keysArray[11] = new Button("Fminor");
            keysArray[12] = new Button("F#major");
            keysArray[13] = new Button("F#minor");

            keysArray[14] = new Button("Gmajor");
            keysArray[15] = new Button("Gminor");
            keysArray[16] = new Button("G#major");
```

```
            keysArray[17] = new Button("G#minor");

            keysArray[18] = new Button("Amajor");
            keysArray[19] = new Button("Aminor");
            keysArray[20] = new Button("A#major");
            keysArray[21] = new Button("A#minor");

            keysArray[22] = new Button("Bmajor");
            keysArray[23] = new Button("Bminor");

            // prosthetoume ta koumpia panw sto panel

            for (int i = 0; i < 24; i++) {
                  keysArray[i].setForeground(new java.awt.Color(102,
0, 102));
                  keysArray[i].setFont(new java.awt.Font(null, 1,
12));
                  keyPad.add(keysArray[i]);

            }
            for (int i = 0; i < keysArray.length; i++) {
                  keysArray[i].addActionListener(new
java.awt.event.ActionListener() {
                        public void
actionPerformed(java.awt.event.ActionEvent evt) {
                              jButton2ActionPerformed(evt);
                        }
                  });
            }

            Label nothing = new Label("");
            Panel labelPanel2 = new Panel();
            labelPanel2.setLayout(new GridLayout(0, 1));
            Label organ = new Label(
                        "  Please Enter which Organ do you want to
play the music:");
            organ.setForeground(new java.awt.Color(102, 0, 102));
            organ.setFont(new java.awt.Font(null, 1, 12));
            // labelPanel.add(organ);

            labelPanel2.add(organ);

            keyOrgan = new Panel();
            keyOrgan.setLayout(new GridLayout(0, 1));
            keysOrgan = new Button[6];

            keysOrgan[0] = new Button("Violin");
            keysOrgan[1] = new Button("Viola");
            keysOrgan[2] = new Button("Cello");
            keysOrgan[3] = new Button("Piano");
            keysOrgan[4] = new Button("Oboe");
            keysOrgan[5] = new Button("Flute");

            // prosthetoume ta koumpia panw sto panel

            for (int i = 0; i < 6; i++) {
                  keysOrgan[i].setForeground(new java.awt.Color(102,
0, 102));
                  keysOrgan[i].setFont(new java.awt.Font(null, 1,
12));
                  keyOrgan.add(keysOrgan[i]);
```

```
            }
            for (int i = 0; i < keysOrgan.length; i++) {
                  keysOrgan[i].addActionListener(new
java.awt.event.ActionListener() {
                        public void
actionPerformed(java.awt.event.ActionEvent evt) {
                              jButton1ActionPerformed(evt);
                        }
                  });
            }

            Label critics = new Label(
                        "Please Choose which criterion you want :
Consecutive same notes: ");
            critics.setForeground(new java.awt.Color(102, 0, 102));
            critics.setFont(new java.awt.Font(null, 1, 12));

            Label contentPanel3 = new Label(
                        "
Consecutive pauses: ");
            contentPanel3.setForeground(new java.awt.Color(102, 0,
102));
            contentPanel3.setFont(new java.awt.Font(null, 1, 12));

            Label contentPanel4 = new Label(
                        "
Intervals: ");
            contentPanel4.setForeground(new java.awt.Color(102, 0,
102));
            contentPanel4.setFont(new java.awt.Font(null, 1, 12));

            Label contentPanel5 = new Label(
                        "
Notes not in scale: ");
            contentPanel5.setForeground(new java.awt.Color(102, 0,
102));
            contentPanel5.setFont(new java.awt.Font(null, 1, 12));

            Label contentPanel6 = new Label(
                        "
Motives: ");
            contentPanel6.setForeground(new java.awt.Color(102, 0,
102));
            contentPanel6.setFont(new java.awt.Font(null, 1, 12));

            Label contentPanel7 = new Label(
                        "
Same notes between: ");
            contentPanel7.setForeground(new java.awt.Color(102, 0,
102));
            contentPanel7.setFont(new java.awt.Font(null, 1, 12));

            Label contentPanel8 = new Label(" ");
            Panel labelPanel7 = new Panel();
            labelPanel7.setLayout(new GridLayout(0, 1));

            labelPanel.add(critics);
            labelPanel.add(contentPanel3);
            labelPanel.add(contentPanel4);
            labelPanel.add(contentPanel5);
```

```
            labelPanel.add(contentPanel6);
            labelPanel.add(contentPanel7);
            // labelPanel7.add(contentPanel8);

            contentPanel21 = new TextField("Y/N");
            contentPanel21.setBackground(new java.awt.Color(102, 0,
102));
            contentPanel21.setForeground(Color.white);

            contentPanel31 = new TextField("Y/N");
            contentPanel31.setBackground(new java.awt.Color(102, 0,
102));
            contentPanel31.setForeground(Color.white);

            contentPanel41 = new TextField("Y/N");
            contentPanel41.setBackground(new java.awt.Color(102, 0,
102));
            contentPanel41.setForeground(Color.white);

            contentPanel51 = new TextField("Y/N");
            contentPanel51.setBackground(new java.awt.Color(102, 0,
102));
            contentPanel51.setForeground(Color.white);

            contentPanel61 = new TextField("Y/N");
            contentPanel61.setBackground(new java.awt.Color(102, 0,
102));
            contentPanel61.setForeground(Color.white);

            contentPanel71 = new TextField("Y/N");
            contentPanel71.setBackground(new java.awt.Color(102, 0,
102));
            contentPanel71.setForeground(Color.white);

            Panel fieldPanel1 = new Panel();
            fieldPanel1.setLayout(new GridLayout(0, 1));

            fieldPanel.add(contentPanel21);
            fieldPanel.add(contentPanel31);
            fieldPanel.add(contentPanel41);
            fieldPanel.add(contentPanel51);
            fieldPanel.add(contentPanel61);
            fieldPanel.add(contentPanel71);

            this.add(contentPanel12, BorderLayout.BEFORE_FIRST_LINE);
            this.setResizable(false);
            contentPanel.add(labelPanel, BorderLayout.NORTH);
            contentPanel.add(fieldPanel, BorderLayout.NORTH);
            contentPanel.add(labelPanel1, BorderLayout.CENTER);
            contentPanel.add(keyPad, BorderLayout.CENTER);
            contentPanel.add(labelPanel2, BorderLayout.SOUTH);
            contentPanel.add(keyOrgan, BorderLayout.WEST);
            // contentPanel.add(labelPanel7, BorderLayout.WEST);
            Panel p1 = new Panel();
            p1.setLayout(new GridLayout(0, 1));

            // add the buttons

            runBtn = new Button("RUN");
            runBtn.addActionListener(this);
            runBtn.setActionCommand("Run");
```

```
            runBtn.setForeground(new java.awt.Color(102, 0, 102));
            runBtn.setFont(new java.awt.Font(null, 1, 12));

            contentPanel.add(runBtn, BorderLayout.NORTH);
            this.add(contentPanel, BorderLayout.AFTER_LAST_LINE);
            this.setResizable(false);

            // display the window
            this.pack();
            this.show();
      }

      public void windowClosing(WindowEvent we) {

            if (we.getSource() == fr) {

                  fr.setVisible(false);
                  super.enable();
                  super.setVisible(true);

            }

            if (we.getSource() == this) {

                  System.exit(0);

            }
      }

      private void jButton2ActionPerformed(java.awt.event.ActionEvent
evt) {
            if (evt.getSource() == keysArray[0])
                  scale = "Cmajor";
            else if (evt.getSource() == keysArray[1])
                  scale = "Cminor";
            else if (evt.getSource() == keysArray[2])
                  scale = "C#major";
            else if (evt.getSource() == keysArray[3])
                  scale = "C#minor";
            else if (evt.getSource() == keysArray[4])
                  scale = "Dmajor";
            else if (evt.getSource() == keysArray[5])
                  scale = "Dminor";
            else if (evt.getSource() == keysArray[6])
                  scale = "D#major";
            else if (evt.getSource() == keysArray[7])
                  scale = "D#minor";
            else if (evt.getSource() == keysArray[8])
                  scale = "Emajor";
            else if (evt.getSource() == keysArray[9])
                  scale = "Eminor";
            else if (evt.getSource() == keysArray[10])
                  scale = "Fmajor";
            else if (evt.getSource() == keysArray[11])
                  scale = "Fminor";
            else if (evt.getSource() == keysArray[12])
                  scale = "F#major";
            else if (evt.getSource() == keysArray[13])
                  scale = "F#minor";
            else if (evt.getSource() == keysArray[14])
                  scale = "Gmajor";
```

```
            else if (evt.getSource() == keysArray[15])
                  scale = "Gminor";
            else if (evt.getSource() == keysArray[16])
                  scale = "G#major";
            else if (evt.getSource() == keysArray[17])
                  scale = "G#minor";
            else if (evt.getSource() == keysArray[18])
                  scale = "Amajor";
            else if (evt.getSource() == keysArray[19])
                  scale = "Aminor";
            else if (evt.getSource() == keysArray[20])
                  scale = "A#major";
            else if (evt.getSource() == keysArray[21])
                  scale = "A#minor";
            else if (evt.getSource() == keysArray[22])
                  scale = "Bmajor";
            else if (evt.getSource() == keysArray[23])
                  scale = "Bminor";
      }

      private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {

            if (evt.getSource() == keysOrgan[0]) {
                  violin = 1;
                  viola = 0;
                  oboe = 0;
                  flute = 0;
                  piano = 0;
                  tsello = 0;
            } else if (evt.getSource() == keysOrgan[1]) {
                  violin = 0;
                  viola = 1;
                  oboe = 0;
                  flute = 0;
                  piano = 0;
                  tsello = 0;
            } else if (evt.getSource() == keysOrgan[2]) {
                  violin = 0;
                  viola = 0;
                  oboe = 0;
                  flute = 0;
                  piano = 0;
                  tsello = 1;
            } else if (evt.getSource() == keysOrgan[3]) {
                  violin = 0;
                  viola = 0;
                  oboe = 0;
                  flute = 0;
                  piano = 1;
                  tsello = 0;
            } else if (evt.getSource() == keysOrgan[4]) {
                  violin = 0;
                  viola = 0;
                  oboe = 1;
                  flute = 0;
                  piano = 0;
                  tsello = 0;
            } else if (evt.getSource() == keysOrgan[5]) {
                  violin = 0;
                  viola = 0;
```

```
                  oboe = 0;
                  flute = 1;
                  piano = 0;
                  tsello = 0;
            }
      }

      public void windowActivated(WindowEvent we) {
      };

      public void windowClosed(WindowEvent we) {
      };

      public void windowDeactivated(WindowEvent we) {
      };

      public void windowIconified(WindowEvent we) {
      };

      public void windowDeiconified(WindowEvent we) {
      };

      public void windowOpened(WindowEvent we) {
      };

      public void actionPerformed(ActionEvent ae) {

            try {
                  if (ae.getSource() == runBtn) {
                        String organ = "";
                        int popSize =
Integer.valueOf(populationSize.getText())
                                    .intValue();
                        int genNum =
Integer.valueOf(numIterations.getText())
                                    .intValue();
                        double fitNum =
Double.valueOf(fitnessVal.getText())
                                    .doubleValue();
                        int metrNum =
Integer.valueOf(metersVal.getText()).intValue();
                        String k1 =
String.valueOf(contentPanel21.getText());
                        String k2 =
String.valueOf(contentPanel31.getText());
                        String k3 =
String.valueOf(contentPanel41.getText());
                        String k4 =
String.valueOf(contentPanel51.getText());
                        String k5 =
String.valueOf(contentPanel61.getText());
                        String k6 =
String.valueOf(contentPanel71.getText());

                        if ((popSize > 0 && popSize < 200)
                                    && (genNum > 0 && genNum < 200)
&& (fitNum > 0)
                                    && (metrNum > 0) &&
(k1.equals("Y") || k1.equals("N"))
                                    && (k2.equals("Y") ||
k2.equals("N"))
```

```
                                    && (k3.equals("Y") ||
k3.equals("N"))
                                    && (k4.equals("Y") ||
k4.equals("N"))
                                    && (k5.equals("Y") ||
k5.equals("N"))
                                    && (k6.equals("Y") ||
k6.equals("N"))) {
                              int ch1, ch2, ch3, ch4, ch5, ch6;
                              if (k1.equals("Y"))
                                    ch1 = 1;
                              else
                                    ch1 = 0;

                              if (k2.equals("Y"))
                                    ch2 = 1;
                              else
                                    ch2 = 0;

                              if (k3.equals("Y"))
                                    ch3 = 1;
                              else
                                    ch3 = 0;

                              if (k4.equals("Y"))
                                    ch4 = 1;
                              else
                                    ch4 = 0;

                              if (k5.equals("Y"))
                                    ch5 = 1;
                              else
                                    ch5 = 0;

                              if (k6.equals("Y"))
                                    ch6 = 1;
                              else
                                    ch6 = 0;

                              run_algorithm(popSize, metrNum, scale,
genNum, fitNum, ch1,
                                          ch2, ch3, ch4, ch5, ch6);
                        } else {

                              JOptionPane.showMessageDialog(this,
                                          "Invalid Input(s). Please
try again",
                                          "Error Message",
JOptionPane.ERROR_MESSAGE);
                              super.enable();

                        }

                  }
            } catch (Exception ex) {

                  JOptionPane.showMessageDialog(this,
                              "Invalid Input1(s). Please try again",
"Error Message",
                              JOptionPane.ERROR_MESSAGE);
                  super.enable();
```

```
            }

      }

      public static void main(String[] args) throws IOException {
            new genetic_algorithm();
      }

      // Execute genetic algorithm
      static void run_algorithm(int population_size, int num_of_bars,
                  String scale, int loop, double percentage, int kr1,
int kr2,
                  int kr3, int kr4, int kr5, int kr6) {

            int iteration = 0, best = 0, sec = 0;
            boolean flag = false;
            int[] parents_position = new int[2];
            Vector<Phrase> population = new Vector<Phrase>();
            Vector<Phrase> population_s = new Vector<Phrase>();
            Vector<Double> fitness_array = new Vector<Double>();
            Vector<Phrase> population_no2 = new Vector<Phrase>();
            Vector<Phrase> population_no2_s = new Vector<Phrase>();
            Phrase[] parent1 = new Phrase[2];
            Phrase[] second_parent1 = new Phrase[2];
            double rand = 0.0;
            Vector<Double> new_fitness_array = new Vector<Double>();

            // Initialise population
            Initialise_variables initialiser = new
Initialise_variables();
            population =
initialiser.initialise_population(population_size,
                        num_of_bars, scale);
            population_s = initialiser.second_voice_return();

            // Evaluate the fitness for every melody

      fitness_array.addAll(Evaluater.evaluate_fitness(population,
                        population_s, num_of_bars, scale, 4.0, kr1,
kr2, kr3, kr4, kr5,
                        kr6));

            // Terminator
            Terminator termination_algorithm = new Terminator();

            flag = termination_algorithm.terminator_algorithm(loop,
percentage,
                        fitness_array, iteration);

            // Ean vrike termatise apo tin prwti fora 8a proxwrisei
gia na
            // parousiasei ta apotelesmata
            if (flag == true) {
            }

            else {
                  // Efoson den termatise o algorithmos

                  while (flag == false) {
```

```
                        iteration++;

                        // Select new population
                        Survivor new_population = new Survivor();

                        if (iteration == 1) {

      population_no2.addAll(new_population.choose_survivor(
                                          population, population_s,
fitness_array,
                                          population_size));
                              population_no2_s.addAll(new_population

      .return_new_population_second());
                              Survivor n1 = new Survivor();

      new_fitness_array.addAll(n1.return_fitness(population_no2,
                                          fitness_array,
population_size));
                        }

                        // Evaluate the fitness for every
                        // melody*********************************

                        Vector<Double> new_fitness_array2 = new
Vector<Double>();

      new_fitness_array2.addAll(Evaluater.evaluate_fitness(
                                    population_no2, population_no2_s,
num_of_bars, scale,
                                    4.0, kr1, kr2, kr3, kr4, kr5,
kr6));

                        // Parents selection(melodies with 2 bigger
fitness)
                        parents_position[0] = 0;
                        parents_position[1] = 1;

                        rand = Math.random();
                        if (rand >= 0.5) {
                              sec = 0;
                              parent1 =
Crossover.execute_crossover(population_no2
                                          .get(parents_position[0]),
population_no2
                                          .get(parents_position[1]),
num_of_bars, 4, scale,
                                          sec);

                              sec = 1;
                              second_parent1 =
Crossover.execute_crossover(

      population_no2_s.get(parents_position[0]),

      population_no2_s.get(parents_position[1]),
                                          num_of_bars, 4, scale,
sec);
                        } else {

                              // Mutate
```

```
                              Mutate after_mutation = new Mutate();
                              parent1[0] =
after_mutation.mutate_function(population_no2
                                    .get(parents_position[0]),
population_no2_s
                                    .get(parents_position[0]),
scale);
                              second_parent1[0] =
after_mutation.return_offspring2();
                              parent1[1] =
after_mutation.mutate_function(population_no2
                                    .get(parents_position[1]),
population_no2_s
                                    .get(parents_position[1]),
scale);
                              second_parent1[1] =
after_mutation.return_offspring2();
                        }
                        Vector<Phrase> population_3 = new
Vector<Phrase>();
                        Vector<Phrase> population_3_s = new
Vector<Phrase>();
                        Vector<Double> fitness_array_3 = new
Vector<Double>();

                        for (int i = 0; i < population.size(); i++) {

      population_3.add(population_no2.get(i));

      population_3_s.add(population_no2_s.get(i));

      fitness_array_3.add(new_fitness_array2.get(i));
                        }
                        population_3.set(0, parent1[0]);
                        population_3_s.set(0, second_parent1[0]);
                        fitness_array_3.set(0,
new_fitness_array2.get(0));
                        population_3.set(1, parent1[1]);
                        population_3_s.set(1, second_parent1[1]);
                        fitness_array_3.set(1,
new_fitness_array2.get(1));

                        // Evaluate the fitness for every melody
                        Vector<Double> new_fitness_array4 = new
Vector<Double>();

      new_fitness_array4.addAll(Evaluater.evaluate_fitness(
                              population_3, population_3_s,
num_of_bars, scale, 4.0,
                              kr1, kr2, kr3, kr4, kr5, kr6));

                        // Terminator
                        flag =
termination_algorithm.terminator_algorithm(loop,
                              percentage, new_fitness_array4,
iteration);

                        if (flag == false) {
                              Vector<Phrase> popul_no2 = new
Vector<Phrase>();
```

```
                              Vector<Phrase> popul_no2_s = new
Vector<Phrase>();

      popul_no2.addAll(new_population.choose_survivor(
                                          population_3,
population_3_s, new_fitness_array4,
                                          population_size));
                              popul_no2_s.addAll(new_population

      .return_new_population_second());
                              for (int i = 0; i <
population_no2.size(); i++) {
                                    population_no2.set(i,
popul_no2.get(i));
                                    population_no2_s.set(i,
popul_no2_s.get(i));
                              }

                              Survivor n2 = new Survivor();
                              for (int i = 0; i <
population_no2.size(); i++)
                                    new_fitness_array2.set(i,
n2.return_fitness(
                                                population_3,
new_fitness_array4,

      population_size).get(i));

                        } else {
                              for (int i = 0; i <
population_3.size(); i++) {

      population_4.add(population_3.get(i));

      population_4_s.add(population_3_s.get(i));

      fitness_array_4.add(new_fitness_array4.get(i));
                              }
                        }
                  }
            }

            boolean execute_iteration;

            execute_iteration = Terminator.execute_iteration(loop,
iteration);
            int result;
            if (execute_iteration) {
                  if (iteration == 0)

                        result =
Terminator.solution_no2(fitness_array);
                  else
                        result =
Terminator.solution_no2(fitness_array_4);

                  if (iteration == 0)
                        best =
Terminator.solution_no2(fitness_array);
                  else
```

```
                        best =
Terminator.solution_no2(fitness_array_4);

                  Part p3 = new Part();
                  Part s3 = new Part();
                  Part p2 = new Part();

                  if (violin == 1) {
                        p3 = new Part("Melody Part", 40, 0);
                        s3 = new Part("Second Part", 40, 0);
                        p2 = new Part("Melody Part", 40, 0);
                  } else if (viola == 1) {
                        p3 = new Part("Melody Part", 41, 0);
                        s3 = new Part("Second Part", 41, 0);
                        p2 = new Part("Melody Part", 41, 0);
                  } else if (piano == 1) {
                        p3 = new Part("Melody Part", 3, 0);
                        s3 = new Part("Second Part", 3, 0);
                        p2 = new Part("Melody Part", 3, 0);
                  } else if (tsello == 1) {
                        p3 = new Part("Melody Part", 42, 0);
                        s3 = new Part("Second Part", 42, 0);
                        p2 = new Part("Melody Part", 42, 0);
                  } else if (flute == 1) {
                        p3 = new Part("Melody Part", 73, 0);
                        s3 = new Part("Second Part", 73, 0);
                        p2 = new Part("Melody Part", 73, 0);
                  } else if (oboe == 1) {
                        p3 = new Part("Melody Part", 68, 0);
                        s3 = new Part("Second Part", 68, 0);
                        p2 = new Part("Melody Part", 68, 0);
                  }

                  if (iteration == 0) {
                        p3.addPhrase(population.get(result));
                        s3.addPhrase(population_s.get(result));
                  } else {
                        p3.addPhrase(population_4.get(result));
                        s3.addPhrase(population_4_s.get(result));
                  }

                  s.setTimeSignature(4, 4);
                  p3.setVolume(MEZZO_PIANO);
                  s3.setVolume(PIANO);
                  p2.setVolume(PIANO);
                  s.addPart(p3);
                  s.addPart(s3);
                  View.notate(s, 120, 425);

            } else {
                  boolean execute_percent;
                  if (iteration == 0)
                        execute_percent =
Terminator.execute_percent(fitness_array,
                                    percentage);
                  else
                        execute_percent =
Terminator.execute_percent(fitness_array_4,
                                    percentage);

                  if (execute_percent) {
```

```
                        if (iteration == 0)

                              result =
Terminator.solution(fitness_array, percentage);
                        else

                              result =
Terminator.solution(fitness_array_4, percentage);
                        if (iteration == 0)
                              best =
Terminator.solution_no2(fitness_array);
                        else
                              best =
Terminator.solution_no2(fitness_array_4);
                        Part p3 = new Part();
                        Part s3 = new Part();
                        Part p2 = new Part();

                        if (violin == 1) {
                              p3 = new Part("Melody Part", 40, 0);
                              s3 = new Part("Second Part", 40, 0);
                              p2 = new Part("Melody Part", 40, 0);
                        } else if (viola == 1) {
                              p3 = new Part("Melody Part", 41, 0);
                              s3 = new Part("Second Part", 41, 0);
                              p2 = new Part("Melody Part", 41, 0);
                        } else if (piano == 1) {
                              p3 = new Part("Melody Part", 3, 0);
                              s3 = new Part("Second Part", 3, 0);
                              p2 = new Part("Melody Part", 3, 0);
                        } else if (tsello == 1) {
                              p3 = new Part("Melody Part", 42, 0);
                              s3 = new Part("Second Part", 42, 0);
                              p2 = new Part("Melody Part", 42, 0);
                        } else if (flute == 1) {
                              p3 = new Part("Melody Part", 73, 0);
                              s3 = new Part("Second Part", 73, 0);
                              p2 = new Part("Melody Part", 73, 0);
                        } else if (oboe == 1) {
                              p3 = new Part("Melody Part", 68, 0);
                              s3 = new Part("Second Part", 68, 0);
                              p2 = new Part("Melody Part", 68, 0);
                        }
                        s.setTimeSignature(4, 4);

                        if (iteration == 0) {
                              p3.addPhrase(population.get(result));
                              s3.addPhrase(population_s.get(result));
                        } else {
                              p3.addPhrase(population_4.get(result));

      s3.addPhrase(population_4_s.get(result));
                        }

                        p3.setVolume(MEZZO_PIANO);
                        s3.setVolume(PIANO);
                        p2.setVolume(PIANO);
                        s.addPart(p3);
                        s.addPart(s3);
                        View.notate(s, 120, 425);
                  }
```

```
            }
      }
}



/*
 * Onoma: Tsokkou Panayiota
 * A.T: 955953
 * Onoma Arxeiou: Evaluater
 */

//Libraries
import java.util.*;

import jm.music.data.Note;
import jm.music.data.Phrase;

public class Evaluater extends Initialise_variables {

      static int it = 0;
      static double fitness2 = 0;

      // constructor
      Evaluater() {
      }

      public static Vector<Double> evaluate_fitness(Vector<Phrase>
population,
                  Vector<Phrase> population_s, int num_of_bars,
String scale,
                  double rythm1, int kr1, int kr2, int kr3, int kr4,
int kr5, int kr6) {

            int k = 0;
            double sum = 0.0, x = 0.0;
            Vector<Double> fitness = new Vector<Double>();// pinakas
pou periexei
            // ta
            // fitness ka8e melwdias
            for (int i = 0; i < population.size(); i++) {
                  Vector phrase_notes = new Vector();
                  Enumeration notes;
                  Vector phrase_notes2 = new Vector();
                  Enumeration notes2;

                  k = 0;
                  phrase_notes = population.get(i).getNoteList();
                  phrase_notes2 = population_s.get(i).getNoteList();

                  sum = 0.0;
                  x = 0.0;
                  int[] pitches = new int[phrase_notes.size()];
                  double[] rythm = new double[phrase_notes.size()];
                  int[] pitches2 = new int[phrase_notes2.size()];
                  double[] rythm2 = new double[phrase_notes2.size()];

                  fitness.add(i, 0.0);
                  notes = phrase_notes.elements();
```

```
                notes2 = phrase_notes2.elements();

                while (notes.hasMoreElements()) {// pernoume tis
notes ka8e melwdias
                      Note a = (Note) notes.nextElement();
                      pitches[k] = a.getPitch();
                      rythm[k] = a.getRhythmValue();
                      k++;
                }
                k = 0;
                while (notes2.hasMoreElements()) {// pernoume tis
notes ka8e
                      // melwdias
                      Note a = (Note) notes2.nextElement();
                      pitches2[k] = a.getPitch();
                      rythm2[k] = a.getRhythmValue();
                      k++;
                }

                // Klisi sunartisis gia to kritirio an uparxoun
idies notes
                // sunexomena
                if (kr1 == 1) {
                      x = similar_notes(pitches, rythm);
                      x = x * 0.5;
                      sum += x;
                      x = similar_notes(pitches2, rythm2);
                      x = x * 0.5;
                      sum += x;
                }

                // Klisi sunartisis gia to kritirio an uparxoun
sunexomenes pauseis
                if (kr2 == 1) {
                      x = similar_rest(pitches, rythm,
fitness.get(i));
                      x = x * 0.5;
                      sum += x;
                      x = similar_rest(pitches2, rythm2,
fitness.get(i));
                      x = x * 0.5;
                      sum += x;
                }

                // Klisi sunartisis gia to kritirio an uparxoun
epitrepta kai mi
                // epitrepta diastimata se ka8e melwdia
                if (kr3 == 1) {
                      x = intervals_oktave(pitches);
                      x = x * 0.1;
                      sum += x;
                      sum += fitness2;
                }

                // Klisi sunartisis gia to kritirio an uparxoun
notes stin melodia
                // oi opoies den anikoun stin arxiki klimaka
                if (kr4 == 1) {
                      x = control_notes(pitches, scale);
                      x = x * 0.2;
                      sum += x;
```

```
                        x = control_notes(pitches2, scale);
                        x = x * 0.2;
                        sum += x;
                  }

                  // Klisi sunartisis gia to kritirio an uparxoun
motiva stin melwdia
                  if (kr5 == 1) {
                        x = motive(pitches, rythm, rythm1,
num_of_bars);
                        x = x * 0.3;
                        if (x != 0)
                              System.out.println("!!!!!!!!!!!11");

                        sum += x;
                  }

                  // Klisi sunartisis gia to kritirio an uparxoun
notes stin melwdia k
                  // stin 2i fori p einai oi idies i apexoun diastima
3is i oktavas i
                  // diastima 5is
                  if (kr6 == 1) {
                        x = first_second(pitches, pitches2, scale,
rythm);
                        x = x * 0.3;
                        sum += x;
                  }

                  fitness.set(i, sum);
            }

            return fitness;

      }

      // Kritirio: an uparxoun perissoteres apo 3 fores i idia nota
stin
      // melwdia afaireitai pososto
      static double similar_notes(int[] pitches, double[] rythm) {

            int j = 0;
            int fitness = 0;
            int num_pos = 0;

            for (j = 0; j < pitches.length - 3; j++) {

                  if (pitches[j] > 0
                              && (pitches[j] == pitches[j + 1]
                                          && pitches[j + 1] ==
pitches[j + 2] && pitches[j] > 0)) {
                        num_pos++;
                  }
            }
            fitness = (100 * (pitches.length - num_pos)) /
pitches.length;

            return fitness;
      }

      // Kritirio: an uparxoun 3 sunexomenes fores pausi stin
```

```
	// melwdia afaireitai pososto
	static double similar_rest(int[] pitches, double[] rythm,
double start_fit) {

		int j = 0;
		double fitness = 0.0;
		int num_pos = 0;

		for (j = 0; j < pitches.length - 3; j++) {

			if ((pitches[j] < 0 && pitches[j + 1] < 0 &&
pitches[j + 2] < 0)
					|| (pitches[j] < 0 && pitches[j + 1] <
0 && rythm[j] == 2.0 && (rythm[j + 1] == 2.0 || rythm[j + 1] == 1.0))
					|| (pitches[j] < 0 && pitches[j + 1] <
0 && rythm[j] == 1.0 && (rythm[j + 1] == 2.0))) {
				num_pos++;

			}
		}
		fitness = (100 * (pitches.length - num_pos)) /
pitches.length;

		return fitness;
	}

	// Kritirio: an uparxoun diastimata tis oktavas stin melwdia
tote 8a
	// pros8etoume pososto stin melwdia
	static double intervals_oktave(int[] pitches) {

		int j = 0;
		double fitness = 0.0;
		int result = 0, num_pos = 0, num_pos2 = 0;

		for (j = 0; j < pitches.length - 1; j++) {
			if (pitches[j + 1] < 0)// an einai pausi 8a ele3ei
me tin me8epomeni
			// nota
			{
				result = pitches[j] - pitches[j + 2];
				j++;
			} else
				result = pitches[j] - pitches[j + 1];

			if (result < 0)
				result *= -1;

			// elegxos an duo sunexomenes notes apexoun
diastima mias oktavas
			// akrivws tote prostithetai pososto
			if (result == 12) {
				num_pos2++;
			} else {
				// elegxos an duo sunexomenes notes apexoun
diastima megalutero
				// tis
				// mias oktavas tote afaireitai pososto
				if (result > 12) {
					num_pos++;
				}
```

```
                  }
            }
            fitness = (100 * (pitches.length - num_pos)) /
pitches.length;
            fitness2 = (100 * num_pos2) / pitches.length;

            return fitness;

      }

      // Kritirio: an uparxoun motiva stin melwdia tote pros8etoume
pososto
      static double motive(int[] pitches, double[] rythm, double
rythm1,
                  int num_of_bars) {

            double fitness = 0.0, beatCounter = 0.0;
            int i = 0, k = 0, n = 0, count = 0, counter = 0;
            int[] metres = new int[num_of_bars];// thesi telous gia
ka8e metro
            int[] metres2 = new int[num_of_bars];// # notwn se ka8e
metro
            int[] metres3 = new int[num_of_bars];// thesi arxis gia
ka8e metro

            // Briskoume tis thesis p allazei metro
            for (i = 0; i < num_of_bars; i++) {

                  beatCounter = 0.0;

                  beatCounter += rythm[k];
                  k++;

                  while (beatCounter != 4.0) {
                        beatCounter += rythm[k];
                        k++;
                  }
                  metres[i] = k - 1;
            }

            // Briskoyme poses notes periexei to ka8e metro
            for (i = 0; i < num_of_bars; i++) {
                  if (i == 0)
                        metres3[i] = 0;
                  else
                        metres3[i] = metres[i - 1] + 1;
                  metres2[i] = metres[i] - metres3[i] + 1;
            }

            for (i = 0; i < num_of_bars - 2; i++) {
                  if (metres2[i] == metres2[num_of_bars - 2]) {// an
exoun to idio
                        // mikos ta duo
                        // metra
                        count = metres3[num_of_bars - 2];
                        counter = 0;

                        for (n = metres3[i]; n <= metres[i]; n++) {
                              if (pitches[n] == pitches[count] + 7)
                                    counter++;
                              else
```

```
                                    break;
                              count++;
                        }
                        if (counter == metres2[i]) {
                              fitness += 50;
                        }
                  }
            }

            return fitness;
      }

      // Kritirio an uparxoun notes stin melwdia oi opoies den
anikoun stin
      // epilegmeni klimaka
      static double control_notes(int[] pitches, String scale) {

            int j = 0;
            double fitness = 0.0;
            int[] scale1 = new int[7];
            int flag = 0, num_pos = 0;

            if (scale.equals("Cmajor") || scale.equals("C#major")
                        || scale.equals("Dmajor") ||
scale.equals("Emajor")
                        || scale.equals("Fmajor") ||
scale.equals("F#major")
                        || scale.equals("Gmajor") ||
scale.equals("Amajor")
                        || scale.equals("Bmajor") ||
scale.equals("Cbmajor")
                        || scale.equals("Dbmajor") ||
scale.equals("Gbmajor")
                        || scale.equals("Abmajor") ||
scale.equals("Ebmajor")
                        || scale.equals("Bbmajor"))
                  scale1 = notes_for_scale_major(pitches[0]);
            else
                  scale1 = notes_for_scale_minor(pitches[0]);

            for (int i = 0; i < pitches.length; i++) {

                  flag = 0;

                  for (j = 0; j < scale1.length; j++) {
                        if ((pitches[i] > 0)
                                    && (pitches[i] == scale1[j]
                                                || pitches[i] ==
scale1[j] + 12 || pitches[i] == scale1[j] + 24)) {

                              flag = 1;
                              continue;

                        }
                  }
                  if (flag == 0) {// an den uparxei i sigkekrimeni
nota stin
                        // epilegmeni klimaka
                        num_pos++;
                  }
            }
```

```
            fitness = (100 * (pitches.length - num_pos)) /
pitches.length;

            return fitness;
      }

      // Kritirio an oi notes tis protis fwnis apexoun diastima 3is i
5is apo tis
      // notes tis deuteris fwnis
      static double first_second(int pitches_first[], int
pitches_second[],
                  String scale, double rythm[]) {
            double fitness = 0.0;
            int num_pos = 0;

            for (int i = 1; i < pitches_first.length - 4; i++) {

                  // Elegxos gia diastimata ogdois i idia nota
                  if (pitches_first[i] > 0
                              && pitches_second[i] > 0
                              && (pitches_first[i] ==
pitches_second[i]
                                          || pitches_first[i] ==
pitches_second[i] - 12 || pitches_first[i] - 12 ==
pitches_second[i])) {
                        num_pos++;
                  }

                  if (rythm[i] != 0.25)
                        if (pitches_first[i] > 0
                                    && pitches_second[i + 1] > 0
                                    && (pitches_first[i] ==
pitches_second[i + 1]
                                                || pitches_first[i]
== pitches_second[i + 1] - 12 || pitches_first[i] - 12 ==
pitches_second[i + 1])) {
                              num_pos++;
                        }

                  // Elegxos gia diastimata pemptis
                  if (pitches_first[i] > 0
                              && pitches_second[i] > 0
                              && ((pitches_first[i] ==
pitches_second[i] - 7) || (pitches_first[i] - 7 ==
pitches_second[i]))) {
                        num_pos++;
                  }

                  if (rythm[i] != 0.25)
                        if (pitches_first[i] > 0 && pitches_second[i
+ 1] > 0
                                    && (pitches_first[i] ==
pitches_second[i + 1] - 7)
                                    || (pitches_first[i] - 7 ==
pitches_second[i + 1])) {
                              num_pos++;
                        }

                  // Elegxos gia diastimata tritis
                  if (pitches_first[i] == 62 || pitches_first[i] ==
63
```

```
                                || pitches_first[i] == 64 ||
pitches_first[i] == 69
                                || pitches_first[i] == 70 ||
pitches_first[i] == 71
                                || pitches_first[i] == 74 ||
pitches_first[i] == 75
                                || pitches_first[i] == 76) {
                        if (pitches_first[i] > 0 && pitches_second[i]
> 0
                                        && ((pitches_first[i] ==
pitches_second[i] - 3)))// ||
                        // (pitches_first[i]
                        // -
                        // 3
                        // ==
                        // pitches_second[i])))
                        // {
                        {
                                num_pos++;
                                //
System.out.println(pitches_first[i]);
                                //
System.out.println(pitches_second[i]);

                        }
                    } else {
                                if (pitches_first[i] == 65 ||
pitches_first[i] == 66
                                        || pitches_first[i] == 67 ||
pitches_first[i] == 68
                                        || pitches_first[i] == 60 ||
pitches_first[i] == 61
                                        || pitches_first[i] == 72 ||
pitches_first[i] == 77) {
                                if (pitches_first[i] > 0 &&
pitches_second[i] > 0
                                                && ((pitches_first[i] ==
pitches_second[i] - 4)))// ||
                                // (pitches_first[i]
                                // -
                                // 4
                                // ==
                                // pitches_second[i])))
                                // {
                                {
                                        num_pos++;
                                        //
System.out.println(pitches_first[i]);
                                        //
System.out.println(pitches_second[i]);

                                }
                        }
                    }

            }
            // System.out.print("num_pos = ");
            // System.out.println(num_pos);
            fitness = (100 * (pitches_first.length - num_pos))
                        / pitches_first.length;
```

```
	return fitness;
}

// Sunartisi pou periexei tis notes ka8e klimakas
public static int[] notes_for_scale_major(int root) {

	int[] scale = new int[7];

	// Cmajor
	if (root == c4) {
		scale[0] = c4;
		scale[1] = d4;
		scale[2] = e4;
		scale[3] = f4;
		scale[4] = g4;
		scale[5] = a4;
		scale[6] = b4;
	}

	// C#major
	if (root == cs4) {
		scale[0] = df4;
		scale[1] = ef4;
		scale[2] = f4;
		scale[3] = gf4;
		scale[4] = af4;
		scale[5] = bf4;
		scale[6] = c4;
	}

	// Dmajor
	if (root == d4) {
		scale[0] = d4;
		scale[1] = e4;
		scale[2] = fs4;
		scale[3] = g4;
		scale[4] = a4;
		scale[5] = b4;
		scale[6] = cs4;
	}

	// Eflat major
	if (root == ds4) {
		scale[0] = ef4;
		scale[1] = f4;
		scale[2] = g4;
		scale[3] = af4;
		scale[4] = bf4;
		scale[5] = c4;
		scale[6] = d4;
	}

	// Emajor
	if (root == e4) {
		scale[0] = e4;
		scale[1] = fs4;
		scale[2] = gs4;
		scale[3] = a4;
		scale[4] = bs4;
		scale[5] = cs4;
		scale[6] = d4;
```

```
}

// Fmajor
if (root == f4) {
      scale[0] = f4;
      scale[1] = g4;
      scale[2] = a4;
      scale[3] = bf4;
      scale[4] = c4;
      scale[5] = d4;
      scale[6] = e4;
}

// F#major
if (root == fs4) {
      scale[0] = fs4;
      scale[1] = gs4;
      scale[2] = as4;
      scale[3] = b4;
      scale[4] = cs4;
      scale[5] = ds4;
      scale[6] = es4;
}

// Gmajor
if (root == g4) {
      scale[0] = g4;
      scale[1] = a4;
      scale[2] = b4;
      scale[3] = c4;
      scale[4] = d4;
      scale[5] = e4;
      scale[6] = fs4;
}

// a flat major
if (root == gs4) {
      scale[0] = af4;
      scale[1] = bf4;
      scale[2] = c4;
      scale[3] = df4;
      scale[4] = ef4;
      scale[5] = f4;
      scale[6] = g4;
}

// amajor
if (root == a4) {
      scale[0] = a4;
      scale[1] = b4;
      scale[2] = cs4;
      scale[3] = d4;
      scale[4] = e4;
      scale[5] = fs4;
      scale[6] = gs4;
}

// bflat major
if (root == as4) {
      scale[0] = bf4;
      scale[1] = c4;
```

```
            scale[2] = d4;
            scale[3] = ef4;
            scale[4] = f4;
            scale[5] = g4;
            scale[6] = a4;
      }

      // Bmajor
      if (root == b4) {
            scale[0] = b4;
            scale[1] = cs4;
            scale[2] = ds4;
            scale[3] = e4;
            scale[4] = fs4;
            scale[5] = gs4;
            scale[6] = as4;
      }

      // Cbmajor
      if (root == cf4) {
            scale[0] = cf4;
            scale[1] = df4;
            scale[2] = ef4;
            scale[3] = ff4;
            scale[4] = gf4;
            scale[5] = af4;
            scale[6] = bf4;
      }

      // Dbmajor
      if (root == df4) {
            scale[0] = df4;
            scale[1] = ef4;
            scale[2] = f4;
            scale[3] = gf4;
            scale[4] = af4;
            scale[5] = bf4;
            scale[6] = c4;
      }

      // Gbmajor
      if (root == gf4) {
            scale[0] = gf4;
            scale[1] = af4;
            scale[2] = bf4;
            scale[3] = cf4;
            scale[4] = df4;
            scale[5] = ef4;
            scale[6] = f4;
      }

      return scale;
}

static int[] notes_for_scale_minor(int root) {

      int[] scale = new int[7];

      {
            // Cminor
            if (root == c4) {
```

```
      scale[0] = c4;
      scale[1] = d4;
      scale[2] = ef4;
      scale[3] = f4;
      scale[4] = g4;
      scale[5] = af4;
      scale[6] = bf4;
}

// C#minor
if (root == cs4) {
      scale[0] = cs4;
      scale[1] = ds4;
      scale[2] = e4;
      scale[3] = fs4;
      scale[4] = gs4;
      scale[5] = a4;
      scale[6] = b4;
}

// Dminor
if (root == d4) {
      scale[0] = d4;
      scale[1] = e4;
      scale[2] = f4;
      scale[3] = g4;
      scale[4] = a4;
      scale[5] = bf4;
      scale[6] = c4;
}

// D#minor
if (root == ds4) {
      scale[0] = ef4;
      scale[1] = f4;
      scale[2] = gf4;
      scale[3] = af4;
      scale[4] = bf4;
      scale[5] = cf4;
      scale[6] = df4;
}

// Eminor
if (root == e4) {
      scale[0] = e4;
      scale[1] = fs4;
      scale[2] = g4;
      scale[3] = a4;
      scale[4] = b4;
      scale[5] = c4;
      scale[6] = d4;
}

// Fminor
if (root == f4) {
      scale[0] = f4;
      scale[1] = g4;
      scale[2] = af4;
      scale[3] = bf4;
      scale[4] = c4;
      scale[5] = df4;
```

```
      scale[6] = ef4;
}

// F#minor
if (root == fs4) {
      scale[0] = fs4;
      scale[1] = gs4;
      scale[2] = a4;
      scale[3] = b4;
      scale[4] = cs4;
      scale[5] = d4;
      scale[6] = e4;
}

// Gminor
if (root == g4) {
      scale[0] = g4;
      scale[1] = a4;
      scale[2] = bf4;
      scale[3] = c4;
      scale[4] = d4;
      scale[5] = ef4;
      scale[6] = f4;
}

// G#minor
if (root == gs4) {
      scale[0] = gs4;
      scale[1] = as4;
      scale[2] = b4;
      scale[3] = cs4;
      scale[4] = ds4;
      scale[5] = e4;
      scale[6] = fs4;
}

// aminor
if (root == a4) {
      scale[0] = a4;
      scale[1] = b4;
      scale[2] = c4;
      scale[3] = d4;
      scale[4] = e4;
      scale[5] = f4;
      scale[6] = g4;
}

// a#minor
if (root == as4) {
      scale[0] = bf4;
      scale[1] = c4;
      scale[2] = df4;
      scale[3] = ef4;
      scale[4] = f4;
      scale[5] = gf4;
      scale[6] = af4;
}

// Bminor
if (root == b4) {
      scale[0] = b4;
```

```
                        scale[1] = cs4;
                        scale[2] = d4;
                        scale[3] = e4;
                        scale[4] = fs4;
                        scale[5] = g4;
                        scale[6] = a4;
                  }

                  return scale;
            }
      }
}



/*
 * Onoma: Tsokkou Panayiota
 * A.T: 955953
 * Onoma Arxeiou: Initialise_variables
 */

//Libraries
import java.util.Vector;

import jm.JMC;
import jm.music.data.*;
import jm.util.View;

//Initialise melodies, chords and table for population
public class Initialise_variables implements JMC {

      Vector<Phrase> second_voice1 = new Vector<Phrase>();

      // constructor
      public Initialise_variables() {
      }

      public Vector<Phrase> initialise_population(int populationSize,
                  int num_of_bars, String scale) {

            Vector<Phrase> initial_population = new Vector<Phrase>();

            Composer composer = new Composer();

            // Initialise population with melodies
            for (int i = 0; i < populationSize; i++) {

      initial_population.add(composer.compose_phrase(num_of_bars,
scale));
                  second_voice1.add(composer.second_voice);
            }
            return initial_population;
      }

      // epistrefei oles tis deuteres fwnes
      public Vector<Phrase> second_voice_return() {
            return second_voice1;
      }
}
```

```
/*
 * Onoma: Tsokkou Panayiota
 * A.T: 955953
 * Onoma Arxeiou: Composer
 */

//Libraries
import java.util.Enumeration;
import java.util.Random;
import java.util.Vector;

import jm.music.data.CPhrase;
import jm.music.data.Note;
import jm.music.data.Phrase;

public class Mutate {

      static Phrase offspring2;

      // constructor
      Mutate() {
      }

      Phrase mutate_function(Phrase table, Phrase second, String
scale) {

            Vector phrase_notes = new Vector();
            Enumeration notes;
            Vector phrase_notes_s = new Vector();
            Enumeration notes_s;
            Vector chord_phrase_notes = new Vector();
            Enumeration chord_notes;
            int k = 0, klimaka = 0;
            int mut1 = 0, fwni_1i = 0;
            int[] scale1 = new int[7];
            int flag = 1, pr = 0, nx = 0, ps = 0, count = 0,
dekato_ekto = 0;
            Phrase offspring1 = new Phrase();
            int previous = 0, beat = 0;
            offspring2 = new Phrase();

            phrase_notes = table.getNoteList();
            notes = phrase_notes.elements();
            phrase_notes_s = second.getNoteList();
            notes_s = phrase_notes_s.elements();

            int[] pitches = new int[phrase_notes.size()];
            double[] rythm2 = new double[phrase_notes.size()];
            int[] pitches_s = new int[phrase_notes_s.size()];
            double[] rythm2_s = new double[phrase_notes_s.size()];

            // Random generator2 = new Random();
            while (notes.hasMoreElements()) {// pernoume tis notes
ka8e melwdias
                  Note a = (Note) notes.nextElement();
                  pitches[k] = a.getPitch();
                  rythm2[k] = a.getRhythmValue();
```

```
                  k++;
            }
            k = 0;
            while (notes_s.hasMoreElements()) {// pernoume tis notes
ka8e melwdias
                  Note a = (Note) notes_s.nextElement();
                  pitches_s[k] = a.getPitch();
                  rythm2_s[k] = a.getRhythmValue();
                  k++;
            }

            // Briskoume tis notes tis arxikis klimakas
            if (scale.equals("Cmajor") || scale.equals("C#major")
                        || scale.equals("Dmajor") ||
scale.equals("D#major")
                        || scale.equals("Emajor") ||
scale.equals("Fmajor")
                        || scale.equals("F#major") ||
scale.equals("Gmajor")
                        || scale.equals("G#major") ||
scale.equals("Amajor")
                        || scale.equals("A#major") ||
scale.equals("Bmajor")) {
                  klimaka = 1;
                  scale1 =
Evaluater.notes_for_scale_major(pitches[0]);
            } else {
                  klimaka = 0;
                  scale1 =
Evaluater.notes_for_scale_minor(pitches[0]);
            }

            // BRISKOUME TO SHMEIO POU 8a GINEI MUTaTE
            flag = 0;
            for (int i = 1; i < pitches.length; i++) {
                  while (pitches[i] < 0)
                        // efoson einai pausi dn dexomaste gia mutate
                        i++;
                  for (int j = 0; j < scale1.length && flag == 0;
j++) {
                        if (pitches[i] == scale1[j] || pitches[i] ==
scale1[j] + 12)// i
                        // nota
                        // anikei
                        // stin
                        // klimaka
                        {
                              flag = 1;
                        }
                  }
                  if (flag == 0) {// an i nota dn anikei stin klimaka
                        mut1 = i;
                        i = pitches.length;
                  }
                  flag = 0;
            }

            // se periptwsi p dn vre8ike simeio gia na ginei mutate
stin 1i fwni 8a
            // ginei elegxos gia tin 2i fwni
            if (mut1 == 0) {
```

```
                  fwni_1i = 1;
                  // BRISKOUME TO SHMEIO POU 8a GINEI MUTaTE
                  flag = 0;
                  for (int i = 1; i < pitches_s.length; i++) {
                        while (pitches_s[i] < 0)
                              // efoson einai pausi dn dexomaste gia
mutate
                              i++;
                        for (int j = 0; j < scale1.length && flag ==
0; j++) {
                              if (pitches_s[i] == scale1[j]
                                          || pitches_s[i] ==
scale1[j] + 12)// i
                              // nota
                              // anikei
                              // stin
                              // klimaka
                              {
                                    flag = 1;
                              }
                        }
                        if (flag == 0) {// an i nota dn anikei stin
klimaka
                              mut1 = i;
                              i = pitches_s.length;
                        }
                        flag = 0;
                  }
            }

            if (mut1 == 0)// an dn vre8ike nota gia na ginei allagi
oute stin 2i
            // fwni
            {
                  // offspring3 = chord;
                  offspring2 = second;
                  return table;
            }

            count = 0;

            if (fwni_1i == 0) {
                  for (int i1 = 0; i1 < table.size(); i1++) {
                        if (mut1 != i1) {// antigrafi notwn mexri na
f8asoume sto simeio
                              // p 8a ginei mutate
                              offspring1.addNote(table.getNote(i1));

                              if (i1 == 0) {// 1i nota

      offspring2.addNote(second.getNote(count));
                                    count++;
                              } else {
                                    if (rythm2[i1] == 0.25) {
                                          if (dekato_ekto == 0) {

      offspring2.addNote(second.getNote(count));
                                                count++;
                                                dekato_ekto = 1;
                                          } else {
                                                dekato_ekto = 0;
```

```
                                        }
                                    } else {

      offspring2.addNote(second.getNote(count));
                                        count++;

      offspring2.addNote(second.getNote(count));
                                        count++;
                                    }
                                }
                            } else {// simeio p 8a ginei mutate
                                  for (int i11 = 0; i11 < scale1.length;
i11++) {
                                    if ((pitches[mut1 - 1] ==
scale1[i11] || pitches[mut1 - 1] - 12 == scale1[i11])) {// elegxos
                                        // an
                                        // i
                                        // proigoumeni
                                        // nota apo auti
                                        // p 8a ginei
                                        // mutate anikei
                                        // stin arxiki
                                        // klimakas
                                        if ((pitches[mut1 - 1] -
12) == scale1[i11]) {
                                              nx = 1;
                                        } else {
                                              nx = 0;
                                        }
                                        if (i11 != 0)
                                              ps = i11 - 1;
                                        else
                                              ps = i11 + 1;
                                        i11 = scale1.length;
                                    } else if ((pitches[mut1 + 1] ==
scale1[i11] || pitches[mut1 + 1] - 12 == scale1[i11])) {// an
                                        // dn
                                        // anikei
                                        // i
                                        // proigoumeni
                                        // nota
                                        // elegxoume
                                        // tin
                                        // epomeni
                                        // nota
                                        // autis
                                        // p 8a
                                        // ginei
                                        // mutate
                                        if ((pitches[mut1 + 1] -
12) == scale1[i11]) {
                                              pr = 1;
                                        } else {
                                              pr = 0;
                                        }
                                        if (i11 != scale.length())
                                              ps = i11 + 1;
                                        else
                                              ps = i11 - 1;
                                        i11 = scale1.length;
                                    } else {
```

```
                                        // an dn anikei oute i
epomeni oute i proigoumeni
                                        // nota 8a valoume tin
8emelio
                                        ps = 0;// 8emelios
                                  }
                            }

                            Note n;
                            if (nx == 1 || pr == 1) {
                                  n = new Note(scale1[ps] + 12,
table.getNote(mut1)
                                              .getRhythmValue());
                            } else
                                  n = new Note(scale1[ps],
table.getNote(mut1)
                                              .getRhythmValue());
                            offspring1.addNote(n);

                            if (rythm2[mut1] == 0.25) {
                                  if (dekato_ekto == 0) {
                                        Note n1 = new
Note(n.getPitch() + 7, n

      .getRhythmValue() * 2);
                                        offspring2.addNote(n1);
                                        count++;
                                        dekato_ekto = 1;
                                  } else
                                        dekato_ekto = 0;
                            } else {
                                  Note n1 = new Note(n.getPitch() +
7,
                                              n.getRhythmValue() /
2);
                                  offspring2.addNote(n1);
                                  count++;
                                  Note n2 = new Note(n.getPitch(),
n.getRhythmValue() / 2);
                                  offspring2.addNote(n2);
                                  count++;
                            }
                      }
                }
            } else {// periptwsi p 8a ginei mutate stin 2i fwni
                  count = 0;
                  Note n3 = new Note();
                  int i1 = 0;
                  offspring2 = new Phrase();
                  while (mut1 != count) {// antigrafi notwn mexri na
f8asoume sto
                        // simeio
                        // p 8a ginei mutate
                        if (i1 == 0) {// 1i nota
                              offspring1.addNote(table.getNote(i1));


      offspring2.addNote(second.getNote(count));
                              beat = 2;
                              count++;
                              i1++;
```

```
                        } else {
                              if (rythm2[i1] == 0.25) {

      offspring1.addNote(table.getNote(i1));

                                    i1++;

      offspring2.addNote(second.getNote(count));
                                    count++;

      offspring1.addNote(table.getNote(i1));

                                    beat = 2;
                                    i1++;
                              } else {
                                    n3 = table.getNote(i1);

      offspring1.addNote(table.getNote(i1));

                                    i1++;
                                    beat = 1;

      offspring2.addNote(second.getNote(count));
                                    count++;
                                    if (count != mut1) {
                                          beat = 2;

      offspring2.addNote(second.getNote(count));
                                          count++;
                                    } else {
                                          beat = 1;
                                          break;
                                    }
                              }
                        }
                  }
                  // simeio p 8a ginei mutate
                  if (mut1 == count) {
                        if (beat == 1) {
                              n3 = table.getNote(i1 - 1);

                              offspring2.addNote(n3.getPitch(),
n3.getRhythmValue() / 2);

                              count++;
                        } else if (beat == 2) {
                              n3 = table.getNote(i1);
                              offspring1.addNote(table.getNote(i1));

                              Note n6;
                              count++;
                              if (rythm2[i1] == 0.25) {
                                    n6 = new
Note(table.getNote(i1).getPitch() + 7, table

      .getNote(i1).getRhythmValue() * 2);
                                    offspring2.addNote(n6);
                                    dekato_ekto = 1;

                              } else {
```

```
                                    n6 = new
Note(table.getNote(i1).getPitch() + 7, table

      .getNote(i1).getRhythmValue() / 2);
                                    offspring2.addNote(n6);
                                    n6 = new
Note(table.getNote(i1).getPitch(), table

      .getNote(i1).getRhythmValue() / 2);
                                    offspring2.addNote(n6);
                                    count++;
                                    dekato_ekto = 0;
                              }
                              i1++;
                        }
                  }
                  // prosthetoume tis upoloipes notes
                  for (int i2 = i1; i2 < table.length(); i2++) {
                        offspring1.addNote(table.getNote(i2));

                        if (rythm2[i2] == 0.25) {
                              if (dekato_ekto == 0) {

      offspring2.addNote(second.getNote(count));
                                    count++;
                                    dekato_ekto = 1;
                              } else {
                                    dekato_ekto = 0;
                              }
                        } else {

      offspring2.addNote(second.getNote(count));
                              count++;

      offspring2.addNote(second.getNote(count));
                              count++;
                        }
                  }
            }

            return offspring1;
      }

      Phrase return_offspring2() {
            return offspring2;
      }
}
```

```
/*
 * Onoma: Tsokkou Panayiota
 * A.T: 955953
 * Onoma Arxeiou: Terminator
 */

//Libraries
import java.util.Enumeration;
import java.util.Random;
```

```
import java.util.Vector;

import jm.JMC;
import jm.music.data.CPhrase;
import jm.music.data.Note;
import jm.music.data.Part;
import jm.music.data.Phrase;
import jm.music.data.Score;
import jm.util.View;

public class Crossover implements JMC {

	static Phrase[] second_new_parents;
	static CPhrase chord_new_parents;
	static CPhrase chord_new_parents1;
	static int point1 = 0;

	// constructor
	Crossover() {
	}

	// sunartisi pou epistrefei kapoio tuxaio simeio gia na ginei i
diastaurwsi
	private static int get_point(int num_of_bars) {
		int point;
		Random generator2 = new Random();
		point = generator2.nextInt(num_of_bars);
		return point;
	}

	static Phrase[] execute_crossover(Phrase parent1, Phrase
parent2,
			int num_of_bars, int rythm1, String scale, int
flag) {

		Phrase[] new_parents = new Phrase[2];
		second_new_parents = new Phrase[2];
		chord_new_parents = new CPhrase();
		chord_new_parents1 = new CPhrase();

		// Ean i pi8anotita einai mikroteri tou 0 8a ginete
one_point_crossover
		// alliws 8a ginete two_point_crossover
		// if (probabillity < 0.5) {

		new_parents = one_point_crossover(parent1, parent2,
num_of_bars,
				rythm1, scale, flag);

		/*
		 * } else {
		 *
		 * new_parents = two_point_crossover(parent1, parent2,
second1, second2,
		 * chord1, chord2, num_of_bars, rythm1, scale);
		 *
		 * }
		 */
		return new_parents;
	}
```

```
      static Phrase[] one_point_crossover(Phrase parent1, Phrase
parent2,
                  int num_of_bars, int rythm1, String scale, int
flag) {

            Vector phrase_notes = new Vector();
            Enumeration notes;
            Vector phrase_notes2 = new Vector();
            Enumeration notes2;

            int k = 0;
            Phrase[] new_parents = new Phrase[2];
            int i = 0, klimaka = 0;
            double sum = 0.0;
            int k1 = 0;
            int k2 = 0;
            int previous = 0;
            int previous2 = 0;
            Phrase new_p1 = new Phrase(), new_p2 = new Phrase();

            if (flag == 0)
                  point1 = get_point(num_of_bars);
            phrase_notes = parent1.getNoteList();
            notes = phrase_notes.elements();
            phrase_notes2 = parent2.getNoteList();
            notes2 = phrase_notes2.elements();

            int[] pitches = new int[phrase_notes.size()];
            int[] pitches2 = new int[phrase_notes2.size()];
            double[] value = new double[phrase_notes.size()];
            double[] value2 = new double[phrase_notes2.size()];

            while (notes.hasMoreElements()) {// pernoume tis notes
ka8e melwdias
                  Note a = (Note) notes.nextElement();
                  pitches[k] = a.getPitch();
                  value[k] = a.getRhythmValue();
                  k++;
            }

            k = 0;

            while (notes2.hasMoreElements()) {// pernoume tis notes
ka8e melwdias
                  Note b = (Note) notes2.nextElement();
                  pitches2[k] = b.getPitch();
                  value2[k] = b.getRhythmValue();
                  k++;
            }

            i = 0;

            while (i < point1) {
                  sum = 0;
                  sum += value[k1];
                  while (sum <= rythm1) {
                        Note n = new Note(pitches[k1], value[k1]);
                        new_p1.addNote(n);

                        if (scale.equals("Cmajor") ||
scale.equals("C#major")
```

```
                                    || scale.equals("Dmajor") ||
scale.equals("D#major")
                                    || scale.equals("Emajor") ||
scale.equals("Fmajor")
                                    || scale.equals("F#major") ||
scale.equals("Gmajor")
                                    || scale.equals("G#major") ||
scale.equals("Amajor")
                                    || scale.equals("A#major") ||
scale.equals("Bmajor")) {
                              klimaka = 1;
                              if (pitches[k1] < 0)
                                    pitches[k1] = previous;

                              previous = pitches[k1];
                        } else {
                              klimaka = 0;
                              if (pitches[k1] < 0)
                                    pitches[k1] = previous;

                              previous = pitches[k1];
                        }

                        k1++;
                        sum += value[k1];

                  }

                  sum = 0;

                  sum += value2[k2];
                  while (sum <= rythm1) {

                        Note n = new Note(pitches2[k2], value2[k2]);
                        new_p2.addNote(n);

                        if (klimaka == 1) {
                              if (pitches2[k2] < 0)
                                    pitches2[k2] = previous2;

                              previous2 = pitches2[k2];
                        } else {
                              if (pitches2[k2] < 0)
                                    pitches2[k2] = previous2;

                              previous2 = pitches2[k2];
                        }

                        k2++;
                        sum += value2[k2];

                  }
                  i++;
            }

            for (i = k2; i < pitches2.length; i++) {

                  Note n = new Note(pitches2[i], value2[i]);
                  n.setRhythmValue(value2[i]);
                  if (klimaka == 1) {
                        if (pitches2[i] < 0)
```

```
                        pitches2[i] = previous2;

                  previous2 = pitches2[i];
            } else {
                  if (pitches2[i] < 0)
                        pitches2[i] = previous2;

                  previous2 = pitches2[i];
            }

            new_p1.addNote(n);
      }

      for (i = k1; i < pitches.length; i++) {

            Note n = new Note(pitches[i], value[i]);
            n.setRhythmValue(value[i]);
            if (klimaka == 1) {
                  if (pitches[i] < 0)
                        pitches[i] = previous;

                  previous = pitches[i];
            } else {
                  if (pitches[i] < 0)
                        pitches[i] = previous;

                  previous = pitches[i];
            }

            new_p2.addNote(n);
      }

      new_parents[0] = new_p1;
      new_parents[1] = new_p2;
      return new_parents;
   }

   static Phrase[] two_point_crossover(Phrase parent1, Phrase
parent2,
            Phrase second1, Phrase second2, CPhrase chord1,
CPhrase chord2,
            int num_of_bars, int rythm1, String scale) {

      int point1 = 0;
      int point2 = 0;

      Vector phrase_notes = new Vector();
      Enumeration notes;
      Vector phrase_notes2 = new Vector();
      Enumeration notes2;

      Vector second_phrase_notes = new Vector();
      Enumeration second_notes;
      Vector second_phrase_notes2 = new Vector();
      Enumeration second_notes2;

      Vector chord_phrase_notes = new Vector();
      Enumeration chord_notes;
      Vector chord_phrase_notes2 = new Vector();
      Enumeration chord_notes2;
```

```
            int k = 0;
            Phrase[] new_parents = new Phrase[2];
            int i = 0, klimaka = 0;
            double sum = 0.0;
            int k1 = 0;
            int k2 = 0;

            second_new_parents = new Phrase[2];
            chord_new_parents = new CPhrase();
            chord_new_parents1 = new CPhrase();

            // epilogi 2 diaforetikwn simeiwn
            point1 = get_point(num_of_bars);
            point2 = get_point(num_of_bars);
            while (point1 == point2 || point2 < point1) {
                  point1 = get_point(num_of_bars);
                  point2 = get_point(num_of_bars);
            }

            phrase_notes = parent1.getNoteList();

            notes = phrase_notes.elements();

            phrase_notes2 = parent2.getNoteList();

            notes2 = phrase_notes2.elements();

            second_phrase_notes = second1.getNoteList();

            second_notes = second_phrase_notes.elements();

            second_phrase_notes2 = second2.getNoteList();

            second_notes2 = second_phrase_notes2.elements();

            int[] pitches = new int[phrase_notes.size()];
            int[] pitches2 = new int[phrase_notes2.size()];
            double[] value = new double[phrase_notes.size()];
            double[] value2 = new double[phrase_notes2.size()];

            int[] second_pitches = new
int[second_phrase_notes.size()];
            int[] second_pitches2 = new
int[second_phrase_notes2.size()];
            double[] second_value = new
double[second_phrase_notes.size()];
            double[] second_value2 = new
double[second_phrase_notes2.size()];

            while (notes.hasMoreElements()) {// pernoume tis notes
ka8e melwdias
                  Note a = (Note) notes.nextElement();
                  Note b = (Note) second_notes.nextElement();

                  pitches[k] = a.getPitch();
                  second_pitches[k] = b.getPitch();

                  value[k] = a.getRhythmValue();
                  second_value[k] = b.getRhythmValue();

                  k++;
```

```
            }

            k = 0;
            while (notes2.hasMoreElements()) {// pernoume tis notes
ka8e melwdias
                  Note b = (Note) notes2.nextElement();
                  Note b1 = (Note) second_notes2.nextElement();

                  pitches2[k] = b.getPitch();
                  second_pitches2[k] = b1.getPitch();

                  value2[k] = b.getRhythmValue();
                  second_value2[k] = b1.getRhythmValue();

                  k++;
            }

            Phrase new_p1 = new Phrase(), new_p2 = new Phrase();
            Phrase second_new_p1 = new Phrase(), second_new_p2 = new
Phrase();

            i = 0;
            Composer cm = new Composer();
            int ch[] = new int[3];
            int previous = 0, previous2 = 0;

            while (i < point1) {

                  sum = 0;

                  sum += value[k1];

                  while (sum <= rythm1) {
                        Note n = new Note(pitches[k1], value[k1]);
                        Note n1 = new Note(second_pitches[k1],
second_value[k1]);

                        new_p1.addNote(n);
                        second_new_p1.addNote(n1);
                        if (scale.equals("Cmajor") ||
scale.equals("C#major")
                                    || scale.equals("Dmajor") ||
scale.equals("D#major")
                                    || scale.equals("Emajor") ||
scale.equals("Fmajor")
                                    || scale.equals("F#major") ||
scale.equals("Gmajor")
                                    || scale.equals("G#major") ||
scale.equals("Amajor")
                                    || scale.equals("A#major") ||
scale.equals("Bmajor")) {
                              if (pitches[k1] < 0)
                                    pitches[k1] = previous;

                              previous = pitches[k1];
                        } else {
                              if (pitches[k1] < 0)
                                    pitches[k1] = previous;

                              previous = pitches[k1];
                        }
```

```
                        chord_new_parents.addChord(ch, value[k1]);

                        k1++;
                        if (value[k1] != 0.125)
                              sum += value[k1];
                        else
                              sum = sum;
                  }

                  sum = 0;

                  sum += value2[k2];

                  while (sum <= rythm1) {
                        Note n = new Note(pitches2[k2], value2[k2]);
                        Note n1 = new Note(second_pitches2[k2],
second_value2[k2]);

                        if (scale.equals("Cmajor") ||
scale.equals("C#major")
                                    || scale.equals("Dmajor") ||
scale.equals("D#major")
                                    || scale.equals("Emajor") ||
scale.equals("Fmajor")
                                    || scale.equals("F#major") ||
scale.equals("Gmajor")
                                    || scale.equals("G#major") ||
scale.equals("Amajor")
                                    || scale.equals("A#major") ||
scale.equals("Bmajor")) {
                              if (pitches2[k2] < 0)
                                    pitches2[k2] = previous2;

                              previous2 = pitches2[k2];
                        } else {
                              if (pitches2[k2] < 0)
                                    pitches2[k2] = previous2;

                              previous2 = pitches2[k2];
                        }
                        chord_new_parents1.addChord(ch, value2[k2]);

                        new_p2.addNote(n);
                        second_new_p2.addNote(n1);

                        k2++;
                        if (value2[k2] != 0.125)
                              sum += value2[k2];
                        else
                              sum = sum;

                  }

                  i++;
            }

            for (i = point1; i < point2; i++) {
                  sum = 0;

                  sum += value2[k2];
```

```
                  while (sum <= rythm1) {
                        Note n = new Note(pitches2[k2], value2[k2]);
                        Note n1 = new Note(second_pitches2[k2],
second_value2[k2]);

                        if (scale.equals("Cmajor") ||
scale.equals("C#major")
                                    || scale.equals("Dmajor") ||
scale.equals("D#major")
                                    || scale.equals("Emajor") ||
scale.equals("Fmajor")
                                    || scale.equals("F#major") ||
scale.equals("Gmajor")
                                    || scale.equals("G#major") ||
scale.equals("Amajor")
                                    || scale.equals("A#major") ||
scale.equals("Bmajor")) {
                              if (pitches2[k2] < 0)
                                    pitches2[k2] = previous2;

                              previous2 = pitches2[k2];
                        } else {
                              if (pitches2[k2] < 0)
                                    pitches2[k2] = previous2;

                              previous2 = pitches2[k2];
                        }

                        chord_new_parents.addChord(ch, value2[k2]);

                        new_p1.addNote(n);
                        second_new_p1.addNote(n);

                        k2++;
                        System.out.print("k2= ");
                        System.out.println(k2);
                        System.out.println();
                        sum += value2[k2];// #

                  }
            }

            for (i = point1; i < point2; i++) {
                  sum = 0;

                  sum += value[k1];

                  while (sum <= rythm1) {
                        Note n = new Note(pitches[k1], value[k1]);
                        Note n1 = new Note(second_pitches[k1],
second_value[k1]);

                        if (scale.equals("Cmajor") ||
scale.equals("C#major")
                                    || scale.equals("Dmajor") ||
scale.equals("D#major")
                                    || scale.equals("Emajor") ||
scale.equals("Fmajor")
                                    || scale.equals("F#major") ||
scale.equals("Gmajor")
```

```
                                    || scale.equals("G#major") ||
scale.equals("Amajor")
                                    || scale.equals("A#major") ||
scale.equals("Bmajor")) {
                              if (pitches[k1] < 0)
                                    pitches[k1] = previous;

                              previous = pitches[k1];
                        } else {
                              if (pitches[k1] < 0)
                                    pitches[k1] = previous;

                              previous = pitches[k1];
                        }
                        chord_new_parents1.addChord(ch, value[k1]);

                        new_p2.addNote(n);
                        second_new_p2.addNote(n1);

                        k1++;
                        sum += value[k1];

                  }
            }
            if (point2 != num_of_bars - 1) {
                  for (i = k1; i < pitches.length; i++) {

                        Note n = new Note(pitches[i], value[i]);
                        Note n1 = new Note(second_pitches[i],
second_value[i]);

                        n.setRhythmValue(value[i]);
                        n1.setRhythmValue(second_value[i]);

                        if (scale.equals("Cmajor") ||
scale.equals("C#major")
                                    || scale.equals("Dmajor") ||
scale.equals("D#major")
                                    || scale.equals("Emajor") ||
scale.equals("Fmajor")
                                    || scale.equals("F#major") ||
scale.equals("Gmajor")
                                    || scale.equals("G#major") ||
scale.equals("Amajor")
                                    || scale.equals("A#major") ||
scale.equals("Bmajor")) {
                              if (pitches[i] < 0)
                                    pitches[i] = previous;

                              previous = pitches[i];
                        } else {
                              if (pitches[i] < 0)
                                    pitches[i] = previous;

                              previous = pitches[i];
                        }
                        chord_new_parents.addChord(ch, value[i]);

                        new_p1.addNote(n);
                        second_new_p1.addNote(n1);
```

```
                }
            }

            if (point2 != num_of_bars - 1) {
                for (i = k2; i < pitches2.length; i++) {

                    Note n = new Note(pitches2[i], value2[i]);
                    Note n1 = new Note(second_pitches2[i],
second_value2[i]);

                    n.setRhythmValue(value2[i]);
                    n1.setRhythmValue(second_value2[i]);

                    if (scale.equals("Cmajor") ||
scale.equals("C#major")
                            || scale.equals("Dmajor") ||
scale.equals("D#major")
                            || scale.equals("Emajor") ||
scale.equals("Fmajor")
                            || scale.equals("F#major") ||
scale.equals("Gmajor")
                            || scale.equals("G#major") ||
scale.equals("Amajor")
                            || scale.equals("A#major") ||
scale.equals("Bmajor")) {
                        if (pitches2[i] < 0)
                            pitches2[i] = previous2;

                        previous2 = pitches2[i];
                    } else {
                        if (pitches[i] < 0)
                            pitches[i] = previous;

                        previous2 = pitches2[i];
                    }
                    chord_new_parents1.addChord(ch, value2[i]);

                    new_p2.addNote(n);
                    second_new_p2.addNote(n1);

                }
            }

            new_parents[0] = new_p1;
            new_parents[1] = new_p2;

            second_new_parents[0] = second_new_p1;
            second_new_parents[1] = second_new_p2;

            return new_parents;

        }

        public static Phrase[] return_second_parents() {
            return second_new_parents;
        }

        public static CPhrase return_chord_parents() {
            return chord_new_parents;
        }
```

```
	public static CPhrase return_chord_parents1() {
		return chord_new_parents1;
	}
}



/*
 * Onoma: Tsokkou Panayiota
 * A.T: 955953
 * Onoma Arxeiou: Composer
 */

//Libraries
import jm.JMC;
import jm.music.data.*;
import jm.util.*;
import java.util.Vector;

public class Composer implements JMC {

	Vector<Part> p2 = new Vector<Part>();
	public Phrase second_voice;

	// constructor
	public Composer() {
	}

	public Phrase compose_phrase(int num_of_bars, String scale) {
		Phrase p = new Phrase();
		p = compose_melody(num_of_bars, scale);
		return p;
	}

	// Sunartisi gia tin dimiourgia melodiwn me vasi tous kanones
	public Phrase compose_melody(int num_of_bars, String scale) {

		Phrase melodyPhrase = new Phrase();
		second_voice = new Phrase();
		int notes[] = new int[7];
		Vector<Double> rythm_values = new Vector<Double>();
		double length = 0.0, probability = 0.0;
		double beatCounter, beatCounter2;
		Note n = null, n1 = null;
		int root = 0, klimaka = 1, motive = 0;
		root = return_root(scale);

		for (int i = 0; i < num_of_bars; i++) {
			beatCounter = 0.0;
			beatCounter2 = 0.0;
			// dimiourgei tuxaia nota
			int pitch = (int) (Math.random() * 17) + 60;

			while (beatCounter < 4.0) {// efoson einai
mikrotero tis
				// diarkeias tou metrou p edwse o
				// xristis

				length = Math.random();// dinetai tixaia
diarkeia se ka8e nota
```

```
                        if (length > 0.75 && length < 1.0)
                              length = 1.0;// an i pi8anotita einai
megaluteri apo 0.5
                        // tote i a3ia tis notas 8a einai tetarto

                        else if (length <= 0.75 && length > 0.5)
                              length = 0.5;// an i pi8anotita einai
mikroteri apo 0.5 tote
                        // i a3ia tis notas 8a einai ogdoo

                        else if (length <= 0.5 && length > 0.25)
                              length = 2.0;// an i pi8anotita einai
isi me 0.5 tote i a3ia
                        // tis notas 8a einai miso

                        else
                              length = 0.25;// i a3ia tis notas 8a
einai dekato-ekto

                        // Dimiourgia notwn
                        if (i == 0 && beatCounter == 0.0)// an einai
i proti nota tis
                        // melwdias tote 8a tis valoume tin 8emelio
nota tis klimakas
                        // kai 8a einai tetarto
                        {
                              n = new Note(root, CROTCHET);
                              n.setRhythmValue(CROTCHET);

                              melodyPhrase.addNote(n);
                              second_voice.addNote(n); // 2i fwni

                              beatCounter += 1.0;
                              beatCounter2 += 1.0;
                              rythm_values.add(beatCounter2);

                              // dimiourgia sugxordias
                              if (scale.equals("Cmajor") ||
scale.equals("C#major")
                                          || scale.equals("Dmajor")
|| scale.equals("Emajor")
                                          || scale.equals("Fmajor")
                                          || scale.equals("F#major")
                                          || scale.equals("Gmajor")
|| scale.equals("Amajor")
                                          || scale.equals("Bmajor")
                                          || scale.equals("Cbmajor")
                                          || scale.equals("Dbmajor")
                                          || scale.equals("Gbmajor")
                                          || scale.equals("Abmajor")
                                          || scale.equals("Ebmajor")
                                          || scale.equals("Bbmajor"))
{
                                    klimaka = 0;// meizona klimaka
                              } else {
                                    klimaka = 1;// elassona klimaka
                              }
                              // analoga me tin arxiki klimaka
vriskoume tis notes p tin
                              // apoteloun
```

```
                              if (klimaka == 0)
                                    notes =
Evaluater.notes_for_scale_major(root);
                              else
                                    notes =
Evaluater.notes_for_scale_minor(root);

                        } else if (i == num_of_bars - 1)// an
                        // einai
                        // i
                        // teleftaio
                        // metro
                        // tis
                        // melwdias
                        {
                              n = new Note(root, 4.0);
                              n1 = new Note(notes[4],
n.getRhythmValue() / 2);

                              melodyPhrase.addNote(n);
                              second_voice.addNote(n1);// 2i fwni

                              n1 = new Note(root, n.getRhythmValue()
/ 2);
                              second_voice.addNote(n1);// 2i fwni

                              beatCounter += 4.0;
                              beatCounter2 += 4.0;

                        } else {
                              while (length + beatCounter > 4.0) {
                                    length = Math.random();
                                    if (length > 0.75 && length <
1.0)
                                          length = 1.0;// an i
pi8anotita einai megaluteri apo
                                    // 0.5
                                    // tote i a3ia tis notas 8a einai
tetarto

                                    else if (length <= 0.75 && length
> 0.5)
                                          length = 0.5;// an i
pi8anotita einai mikroteri apo
                                    // 0.5 tote
                                    // i a3ia tis notas 8a einai
ogdoo

                                    else if (length <= 0.5 && length
> 0.25)
                                          length = 2.0;// an i
pi8anotita einai isi me 0.5
                                    // tote i a3ia
                                    // tis notas 8a einai miso

                                    else
                                          length = 0.25;// i a3ia tis
notas 8a einai
                                    // dekato-ekto
                              }
```

```
                              double x = Math.random();

                              // dimiourgia proteleutaiou metrou
                              if (i == num_of_bars - 2) {

                                    n = new Note(notes[1], 2.0);
                                    melodyPhrase.addNote(n);
                                    beatCounter += 2.0;

                                    n1 = new Note(n.getPitch() + 7,
CROTCHET);
                                    second_voice.addNote(n1);
                                    n1 = new Note(n.getPitch(),
CROTCHET);
                                    second_voice.addNote(n1);
                                    beatCounter2 += 2.0;

                                    n = new Note(notes[0], 1.0);
                                    melodyPhrase.addNote(n);
                                    beatCounter += 1.0;

                                    n1 = new Note(notes[4], 0.5);
                                    second_voice.addNote(n1);
                                    n1 = new Note(notes[2], 0.5);
                                    second_voice.addNote(n1);
                                    beatCounter2 += 1.0;

                                    n = new Note(notes[4], 1.0);
                                    melodyPhrase.addNote(n);
                                    beatCounter += 1.0;

                                    n1 = new Note(n.getPitch() + 7,
0.5);
                                    second_voice.addNote(n1);
                                    n1 = new Note(n.getPitch(), 0.5);
                                    second_voice.addNote(n1);
                                    beatCounter2 += 1.0;
                              } else {

                                    probability = Math.random();

                                    // dimiourgia motivou
                                    if (probability <= 0.2 && motive
== 0
                                                && beatCounter == 0.0
&& i < num_of_bars - 2) {

                                          n = new Note(notes[5],
2.0);
                                          melodyPhrase.addNote(n);
                                          beatCounter += 2.0;

                                          n1 = new Note(n.getPitch()
+ 7, 1.0);
                                          second_voice.addNote(n1);
                                          beatCounter2 += 1.0;

                                          n1 = new Note(n.getPitch(),
1.0);
                                          second_voice.addNote(n1);
                                          beatCounter2 += 1.0;
```

```
                                        n = new Note(notes[4],
1.0);
                                        melodyPhrase.addNote(n);
                                        beatCounter += 1.0;

                                        n1 = new Note(n.getPitch()
+ 7, 0.5);
                                        second_voice.addNote(n1);
                                        beatCounter2 += 0.5;

                                        n1 = new Note(n.getPitch(),
0.5);
                                        second_voice.addNote(n1);
                                        beatCounter2 += 0.5;

                                        n = new Note(notes[1] + 12,
1.0);
                                        melodyPhrase.addNote(n);
                                        beatCounter += 1.0;

                                        n1 = new Note(n.getPitch()
+ 7, 0.5);
                                        second_voice.addNote(n1);
                                        beatCounter2 += 0.5;

                                        n1 = new Note(n.getPitch(),
0.5);
                                        second_voice.addNote(n1);
                                        beatCounter2 += 0.5;

                                        motive = 1;
                                  } else {

                                        pitch = check(n, x, pitch,
klimaka, root);
                                        n = new Note(pitch,
length);
                                        melodyPhrase.addNote(n);
                                        beatCounter += length;

                                        // 2i fwni
                                        if (length == 2.0) {
                                              pitch = (int)
(Math.random() * 17) + 60;
                                              n1 = new Note(pitch,
1.0);
      second_voice.addNote(n1);

                                              beatCounter2 += 1.0;

                                              pitch = (int)
(Math.random() * 17) + 60;
                                              n1 = new Note(pitch,
1.0);
      second_voice.addNote(n1);

                                              beatCounter2 += 1.0;
```

```
                                        } else if (length == 1.0) {
                                              pitch = (int)
(Math.random() * 17) + 60;
                                              n1 = new Note(pitch,
0.5);
      second_voice.addNote(n1);
                                              beatCounter2 += 0.5;
                                              pitch = (int)
(Math.random() * 17) + 60;
                                              n1 = new Note(pitch,
0.5);
      second_voice.addNote(n1);
                                              beatCounter2 += 0.5;
                                        } else if (length == 0.5) {
                                              pitch = (int)
(Math.random() * 17) + 60;
                                              n1 = new Note(pitch,
0.25);
      second_voice.addNote(n1);// 2i fwni
                                              beatCounter2 += 0.25;
                                              pitch = (int)
(Math.random() * 17) + 60;
                                              n1 = new Note(pitch,
0.25);
      second_voice.addNote(n1);// 2i fwni
                                              beatCounter2 += 0.25;
                                        }
                                        // an einai dekato-ekto
vazoume 3ana dekato-ekto
                                        else {
                                              if (length == 0.25) {
                                                    n = new
Note(pitch, length);
      melodyPhrase.addNote(n);
                                                    beatCounter +=
length;
                                                    pitch = (int)
(Math.random() * 17) + 60;
                                                    n1 = new
Note(pitch, 0.5);
      second_voice.addNote(n1);
                                                    beatCounter2 +=
0.5;
```

```
                                                }
                                          }
                                    }
                              }
                        }

                        pitch = (int) (Math.random() * 17) + 60;
                  }
            }
            return melodyPhrase;
      }

      // epistrefei tin deuteri foni
      Phrase return_second() {
            return second_voice;
      }

      // KaNONES
      int check(Note n, double x, int pitch_note, int klimaka, int
root) {
            int notes[] = new int[7];
            int pitch = 0, flag = 0;

            if (klimaka == 0)
                  notes = Evaluater.notes_for_scale_major(root);
            else
                  notes = Evaluater.notes_for_scale_minor(root);

            // an eixame tin 2i nota tis klimakas 8a akolou8isei
            // i
            // 5i tis
            if (n.getPitch() == notes[1]) {
                  pitch = notes[4];
                  flag = 1;
            }

            // an eixame tin 6i nota tis klimakas 8a akolou8isei
            // i 5i tis
            if (n.getPitch() == notes[5]) {
                  pitch = notes[4];
                  flag = 1;
            }

            // an eixame tin 5i nota tis klimakas 8a akolou8isei
            // eite i 1i tis eite i 6i tis
            if (n.getPitch() == notes[4]) {
                  if (x > 0.5) {
                        pitch = notes[0];
                        flag = 1;
                  } else {
                        pitch = notes[5];
                        flag = 1;
                  }
            } else {
                  // dn 3ekiname pote me tin 6i va8mida!
                  // i 6i va8mida akolou8ei MONO meta apo tin 5i
                  pitch = pitch_note;
                  while (pitch == notes[5]) {
                        pitch = (int) (Math.random() * 17) + 60;
                        flag = 1;
                  }
```

```
	}
	if (flag == 1) {
		return pitch;
	} else {
		return pitch_note;
	}
}

// Sunartisi pou epistrefei tin 8emelio nota mias klimakas
int return_root(String scale) {

	int root = 0;

	if (scale.equals("Cmajor")) {
		root = c4;
	} else if (scale.equals("C#major")) {
		root = cs4;
	} else if (scale.equals("Dmajor")) {
		root = d4;
	} else if (scale.equals("Emajor")) {
		root = e4;
	} else if (scale.equals("Fmajor")) {
		root = f4;
	} else if (scale.equals("F#major")) {
		root = fs4;
	} else if (scale.equals("Gmajor")) {
		root = g4;
	} else if (scale.equals("Amajor")) {
		root = a4;
	} else if (scale.equals("Bmajor")) {
		root = b4;
	} else if (scale.equals("Cbmajor")) {
		root = cf4;
	} else if (scale.equals("Gbmajor")) {
		root = gf4;
	} else if (scale.equals("Dbmajor")) {
		root = df4;
	} else if (scale.equals("Abmajor")) {
		root = af4;
	} else if (scale.equals("Ebmajor")) {
		root = ef4;
	} else if (scale.equals("Bbmajor")) {
		root = bf4;
	} else if (scale.equals("Cminor")) {
		root = c4;
	} else if (scale.equals("C#minor")) {
		root = cs4;
	} else if (scale.equals("Dminor")) {
		root = d4;
	} else if (scale.equals("D#minor")) {
		root = ds4;
	} else if (scale.equals("Eminor")) {
		root = e4;
	} else if (scale.equals("Fminor")) {
		root = f4;
	} else if (scale.equals("F#minor")) {
		root = fs4;
	} else if (scale.equals("Gminor")) {
		root = g4;
	} else if (scale.equals("G#minor")) {
		root = gs4;
```

```
            } else if (scale.equals("Aminor")) {
                  root = a4;
            } else if (scale.equals("A#minor")) {
                  root = as4;
            } else if (scale.equals("Bminor")) {
                  root = b4;
            }
            return root;
      }
}



/*
 * Onoma: Tsokkou Panayiota
 * A.T: 955953
 * Onoma Arxeiou: Survivor
 */

//Libraries
import java.util.Vector;

import jm.music.data.CPhrase;
import jm.music.data.Phrase;

public class Survivor {

      Vector<Phrase> new_population_s;
      Vector<CPhrase> new_population_chord;

      // constructor
      Survivor() {
      }

      Vector<Phrase> choose_survivor(Vector<Phrase> population,
                  Vector<Phrase> second, Vector<Double> fitnessArray,
                  int population_size) {

            Vector<Phrase> new_population = new Vector<Phrase>();

            int i = 0;
            int j = 0;
            Phrase max_population = new Phrase();
            Phrase second_max_population = new Phrase();

            new_population_s = new Vector<Phrase>();

            // Sorting descending fitness
            for (i = 0; i < fitnessArray.size() - 1; i++)
                  for (j = i + 1; j < fitnessArray.size(); j++) {
                        if (fitnessArray.get(j) >=
fitnessArray.get(i)) {

                              max_population = population.get(i);
                              population.set(i, population.get(j));
                              population.set(j, max_population);

                              second_max_population = second.get(i);
                              second.set(i, second.get(j));
                              second.set(j, second_max_population);
```

```
                        }
                  }

            // Choose the first population.length for new population
            for (i = 0; i < population_size; i++) {

                  new_population.add(population.get(i));
                  new_population_s.add(second.get(i));

            }
            return new_population;

      }

      // epistrefei tn neo plithismo oso afora tis deuteres fones
      public Vector<Phrase> return_new_population_second() {
            return new_population_s;

      }

      Vector<Double> return_fitness(Vector<Phrase> population,
                  Vector<Double> fitnessArray, int population_size) {

            int i = 0;
            int j = 0;
            double max = 0;
            Vector<Double> new_fitness = new Vector<Double>();

            // Sorting descending fitness
            for (i = 0; i < fitnessArray.size() - 1; i++)
                  for (j = i + 1; j < fitnessArray.size(); j++) {
                        if (fitnessArray.get(j) >=
fitnessArray.get(i)) {

                              max = fitnessArray.get(i);
                              fitnessArray.set(i,
fitnessArray.get(j));
                              fitnessArray.set(j, max);

                        }
                  }

            // Choose the first population.length for new population
            for (i = 0; i < population_size; i++) {
                  new_fitness.add(fitnessArray.get(i));
            }

            return new_fitness;
      }
}
```

```
import java.util.Vector;

/*
 * Onoma: Tsokkou Panayiota
```

```
 * A.T: 955953
 * Onoma Arxeiou: Terminator
 */

public class Terminator {

	// constructor
	Terminator() {
	}

	// H sunartisi epistrefei true i false an o algorithmos prepei
na termatisei
	// me vasi kapoia kritiria
	boolean terminator_algorithm(int max_iteration, double
percentage,
			Vector<Double> fitnessArray, int iteration) {

		// an o algorithmos exei epanalif8ei oses fores zitise o
xtistis
		if (execute_iteration(max_iteration, iteration))
			return true;

		// an o algorithmos exei to epi8umito fitness function
		if (execute_percent(fitnessArray, percentage))
			return true;

		return false;

	}

	// epistrefei true an o algori8mos exei epanalif8ei oses fores
zitise o
	// xristis
	static boolean execute_iteration(int max_iteration, int
iteration) {

		// an o algorithmos exei epanalif8ei oses fores zitise o
xtistis
		if (iteration == max_iteration)
			return true;
		return false;

	}

	// epistrfei true an epiteux8ike to epi8umito apotelesma stin
sunartisi
	// ikanotitas
	static boolean execute_percent(Vector<Double> fitnessArray,
			double percentage) {

		// an o algorithmos exei to epi8umito fitness function
		for (int i = 0; i < fitnessArray.size(); i++)
			if (percentage <= fitnessArray.get(i))
				return true;

		return false;
	}

	// epistrefei tin kaluteri lusi pou vrike
	static int solution(Vector<Double> fitnessArray, double
percentage) {
```

```
            for (int i = 0; i < fitnessArray.size(); i++)
                  if (percentage <= fitnessArray.get(i)) {
                        return i;
                  }

            return 0;
      }

      // epistrefei tin kaluteri lusi pou vrike mexri auti tin stigmi
      static int solution_no2(Vector<Double> fitnessArray) {

            int max = 0;

            for (int i = 1; i < fitnessArray.size(); i++)

                  if (fitnessArray.get(i) >= fitnessArray.get(max)) {
                        max = i;
                  }

            return max;
      }

}
```