Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**«Εικονική αναπαράσταση εφαρμογής γενετικού αλγορίθμου σε οικοσύστημα οντοτήτων με το Processing»** **“Virtual representation of genetic algorithm application on entities of an ecosystem with Processing”**
Ονοματεπώνυμο Φοιτητή	**Λεβάκης Ιωάννης**
Πατρώνυμο	**Κωνσταντίνος**
Αριθμός Μητρώου	**ΜΠΠΛ/ 17028**
Επιβλέπων	**Αλέπης Ευθύμιος, Αναπληρωτής Καθηγητής**

Ημερομηνία Παράδοσης **Δεκέμβριος 2020**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Αλέπης Ευθύμιος Αναπληρωτής Καθηγητής	Πατσάκης Κωνσταντίνος Επίκουρος Καθηγητής	Βίρβου Μαρία Καθηγήτρια

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΛΗΨΗ (ABSTRACT)

Η παρούσα μεταπτυχιακή διατριβή ερευνά τους μηχανισμούς της εξέλιξης των ειδών στη φύση και μελετά τρόπους ώστε να εφαρμοστούν οι βασικές αρχές της ίδιας της εξελικτικής θεωρίας του Δαρβίνου σε μία προσομοίωση του φυσικού κόσμου ενός οικοσυστήματος υλοποιημένο σε μορφή κώδικα.

Υλοποιείται ένας γενετικός αλγόριθμος σε ένα προγραμματιστικό περιβάλλον με εικονική αναπαράσταση (απεικόνιση) ενός οικοσυστήματος ψεύδο-ζωντανών οντοτήτων στις οποίες εφαρμόζονται εξελικτικές τακτικές για την εύρεση της βέλτιστης λύσης του προβλήματος.

Ασχολούμαστε ειδικότερα με την μοντελοποίηση των οντοτήτων, των διαφόρων συμπεριφορών που τις διέπει, το περιβάλλον στο οποίο δραστηριοποιούνται αλλά και την αποσαφήνιση των μηχανισμών εξέλιξής τους με τον ακριβέστερο δυνατό τρόπο.

Ακόμη εξετάζουμε αναλυτικά τη δομή των γενετικών αλγορίθμων και των βιολογικών μηχανισμών από τους οποίους έχουν εμπνευστεί.

Επιπλέον χρησιμοποιείται το Processing ως εργαλείο για την πραγματοποίηση του έργου για το οποίο εμπεριέχεται σύντομη ανάλυση και οδηγίες σχετικά με την αποτελεσματική χρήση του λογισμικού για την περεταίρω εξοικείωση του αναγνώστη με την προγραμματιστική διεπαφή και τις διάφορες τεχνικές σχεδίασης.

This dissertation investigates the mechanisms of species in nature and studies ways to apply the basic principles of Darwin's own evolutionary theory to a simulation of the physical world of an ecosystem implemented in form of code.

A genetic algorithm is implemented in a programming environment having a virtual representation of an ecosystem comprised of pseudo-living entities in which evolutionary tactics will be applied to in order for the best solution for a certain problem to be found.

We deal in particular with the modeling of entities, the various behaviors that govern them, the environment in which they operate and the clarification of their evolution mechanisms in the most accurate way possible.

We also examine in detail the structure of the genetic algorithms and biological mechanisms from which they are inspired.

In addition we use Processing as a tool to carry out the project for which is included a brief analysis and instructions for the effective use of the software to further familiarize the reader with the programming interface and the various design and behavior techniques.

ΕΙΣΑΓΩΓΗ

Σκοπός της εργασίας είναι η μοντελοποίηση ενός φυσικού συστήματος οντοτήτων σε ένα περιβάλλον προσομοίωσης που προσεγγίζει τις πρακτικές της εξέλιξης και της επιβίωσης των ειδών που υφίσταται στη φύση.

Η προσομοίωση θα αποτελείται αρχικά από σχήματα άψυχων αντικειμένων όπου στη συνέχεια αποκτούν τεχνητή νοημοσύνη και νόημα ζωής δρώντας αυτόνομα ως ζωντανές οντότητες.

Φανταζόμαστε τα αντικείμενά μας σαν ένα υποθαλάσσιο οικοσύστημα οντοτήτων, συγκεκριμένα ένα κοπάδι από ψάρια των οποίων σκοπός είναι να πλοηγηθούν κατά τον βέλτιστο δυνατό τρόπο στον πυθμένα της θάλασσας αποφεύγοντας εμπόδια και διανύοντας την λιγότερη απόσταση προς τον στόχο στην προκειμένη περίπτωση ένα κομμάτι τροφής.

Η ζωή των οντοτήτων μας θα έχει αρχή και τέλος, η κάθε γενιά εφόσον φτάσει στο τέλος της ζωής της ανάλογα με τις διάφορες παραμέτρους που θα ορίσουμε για αυτή θα μεταβιβάζουν τα γονίδιά τους στην επόμενη με σκοπό την εξέλιξη του είδους προς την αποτελεσματικότερη επίτευξη του σκοπού.

Οπότε καλούμαστε να μοντελοποιήσουμε τις οντότητες και τις ιδιαιτερότητές τους (σχήμα, συμπεριφορά), αντίστοιχα τα αντικείμενα των εμποδίων, την απεικόνιση της θάλασσας, τον ίδιο τον γενετικό αλγόριθμο και τους μηχανισμούς που τον διέπουν.

Η κάλυψη αυτού του σκοπού θα εκπληρωθεί δημιουργώντας από το μηδέν έναν γενετικό αλγόριθμο που θα μας δίνει κάθε στιγμή μια βελτιστοποιημένη έκδοση της κάθε προηγούμενης λύσης στο πρόβλημά μας συγκεκριμένα των πιθανών διαδρομών που μπορεί να ακολουθήσει το κοπάδι.

Η υλοποίηση αυτού του αλγορίθμου θα μας επιτρέψει να μιμηθούμε τους μηχανισμούς που υπάρχουν στη φύση επανδρώνοντας στοιχειώδεις έννοιες των μαθηματικών και της φυσικής.

Η κατανόηση των αρχών της φυσικής και των μαθηματικών που ερμηνεύουν τον φυσικό μας κόσμο θα μας βοηθήσει να δημιουργήσουμε και να προσομοιώσουμε τον δικό μας ψηφιακό κόσμο έξυπνων κινούμενων αντικειμένων και σύνθετων συστημάτων αποφάσεων.

Βέβαια δεν θα διερευνήσουμε σε βάθος τις επιστημονικές έννοιες της βιολογίας, αντ 'αυτού θα ρίξουμε μια ματιά σε αυτές και θα αποσπάσουμε τις πληροφορίες που μπορούν να μας φανούν χρήσιμες ώστε να προσομοιώσουμε και να ελέγξουμε αποτελεσματικά τις απρόβλεπτες εξελικτικές ιδιότητες της φύσης στο λογισμικό.

ΚΕΦΑΛΑΙΟ 1: ΓΝΩΡΙΜΙΑ ΜΕ ΤΟ PROCESSING

Εικόνα 1: Λογότυπο του Processing.

Το Processing είναι ένα έργο (project) που ξεκίνησε το 2001 και αντιμετωπίζεται από την κοινότητα ως γλώσσα προγραμματισμού καθώς και ως εργαλείο ανάπτυξης λογισμικού (***I**ntegrated **D**evelopment **E**nvironment*).

Χρησιμοποιείται ευρέως από καθηγητές, μαθητές, καλλιτέχνες, σχεδιαστές, αλλά και ερευνητές για την ανάπτυξη προγραμμάτων σε γραφικό περιβάλλον κάτι που έχει προωθήσει τη μάθηση των εικαστικών τεχνών μέσω του προγραμματισμού.

Πανεπιστήμια καθώς και σχολεία ανά τον κόσμο έχουν εντάξει το Processing ως εργαλείο εκμάθησης προγραμματισμού.

Τα προγράμματα κυρίως προσανατολίζονται στη δημιουργία οπτικών και διαδραστικών στοιχείων. Προγραμματίζοντας με το Processing ο χρήστης αισθάνεται σαν να μαθαίνει να «ζωγραφίζει» σε έναν καμβά χρησιμοποιώντας απλές εντολές.

Δεδομένου της ευκολίας αλλά και της άμεσης προβολής των αποτελεσμάτων του κώδικα, καθιστά την δημιουργία ενός έργου στο Processing εξαιρετικά διασκεδαστικό αλλά και πολύ δημοφιλές σε μαθητευόμενους που κάνουν τα πρώτα τους βήματά στον τομέα της πληροφορικής.

Ακόμη το Processing έχει χρησιμοποιηθεί από χιλιάδες καλλιτέχνες και αρχιτέκτονες για τη δημιουργία των έργων τους πολλά από τα οποία έχουν εκτεθεί σε μουσεία όπως το Μουσείο Μοντέρνας Τέχνης στη Νέα Υόρκη, στο Μουσείο Βικτώριας και Άλμπερτ στο Λονδίνο και σε πολλούς άλλους διακεκριμένους χώρους.

Η πρώτη έκδοση του Processing που κυκλοφόρησε ήταν βασισμένη συντακτικά στη γλώσσα **Java** με επιπρόσθετες γραφικές επιρροές που έχουν βασιστεί σε βιβλιοθήκες όπως το **OpenGL**, ενώ αντικείμενο έμπνευσης για την δημιουργία του Processing υπήρξαν οι προγενέστερες γλώσσες **Logo** και **BASIC**. Πλέον έχουν αναπτυχθεί εκδόσεις που επιτρέπουν την ανάπτυξη λογισμικού και στις γλώσσες **JavaScript**, **Python** και **Ruby** .

Το λογισμικό είναι ανοιχτού κώδικα και παρέχεται δωρεάν για όλους, επίσης είναι συμβατό με τα λειτουργικά συστήματα **Windows** και **Linux.**

Η φύση του ανοιχτού κώδικα του λογισμικού έχει προσελκύσει προγραμματιστές απ' όλο τον κόσμο που διατίθενται να συνεισφέρουν και να συνεργαστούν με ομάδες ανθρώπων για την περεταίρω ανάπτυξη του Processing. Οι συνεργάτες μοιράζονται τα δικά τους προγράμματα με την κοινότητα είτε συνεισφέρουν στην εξέλιξη ενός ήδη υπάρχοντος έργου επεκτείνοντας έτσι τις δυνατότητες του λογισμικού.

Η κοινότητα του Processing έχει καταφέρει να υλοποιήσει περισσότερες από εκατό βιβλιοθήκες, μερικές από τις οποίες έχουν ως σκοπό να βοηθήσουν στην διερεύνηση των θεμάτων όπως υπολογιστική όραση, οπτικοποίηση δεδομένων, προγραμματισμό ηλεκτρονικών συστημάτων.

ΚΕΦΑΛΑΙΟ 2: ΑΡΧΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΣΤΟ PROCESSING

2.1 Προγραμματιστική διεπαφή (Interface):

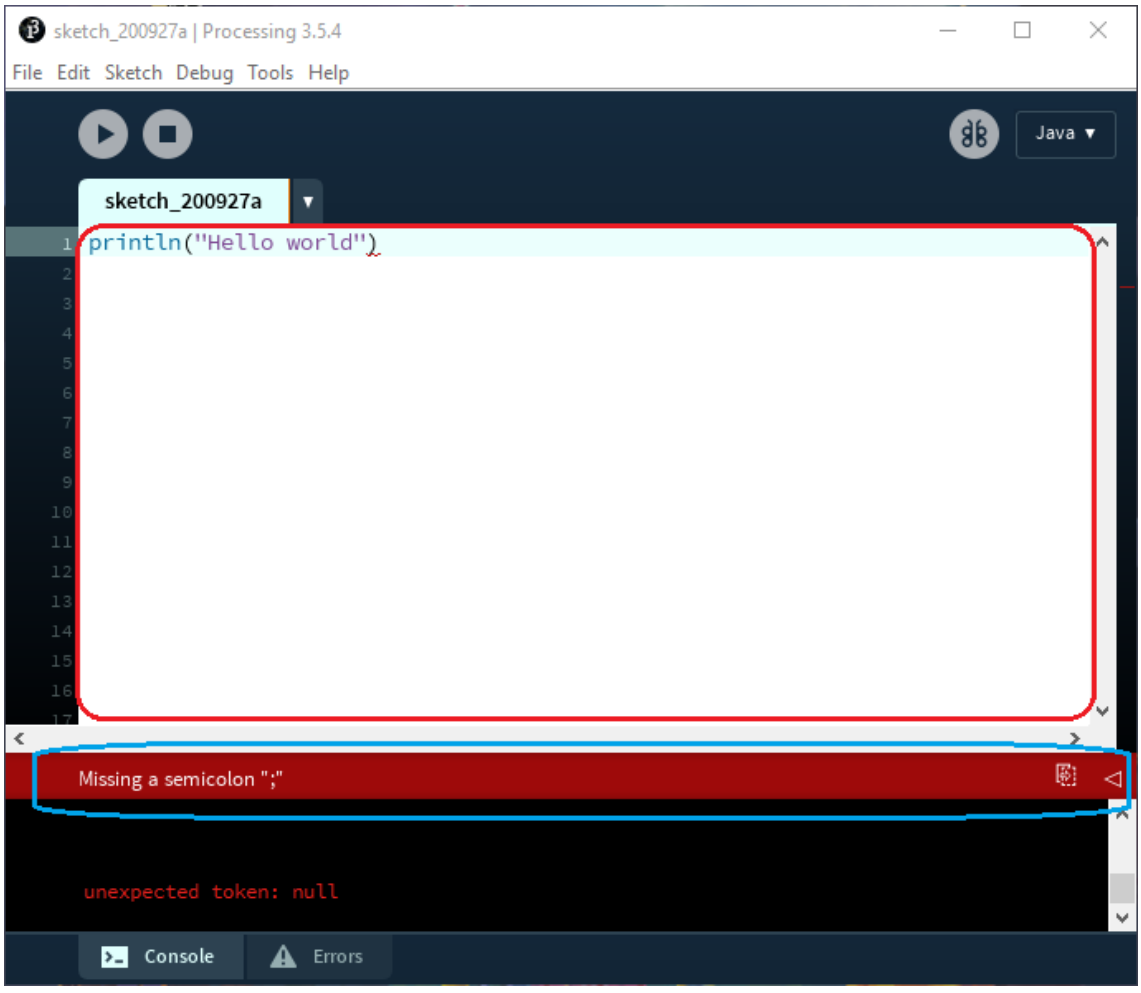
Ξεκινώντας από το περιβάλλον του Processing παρατηρούμε ότι αποτελείται από δύο τμήματα:

- πρώτον έναν επεξεργαστή κειμένου (Text Editor).
- δεύτερον έναν καμβά (Sketch).

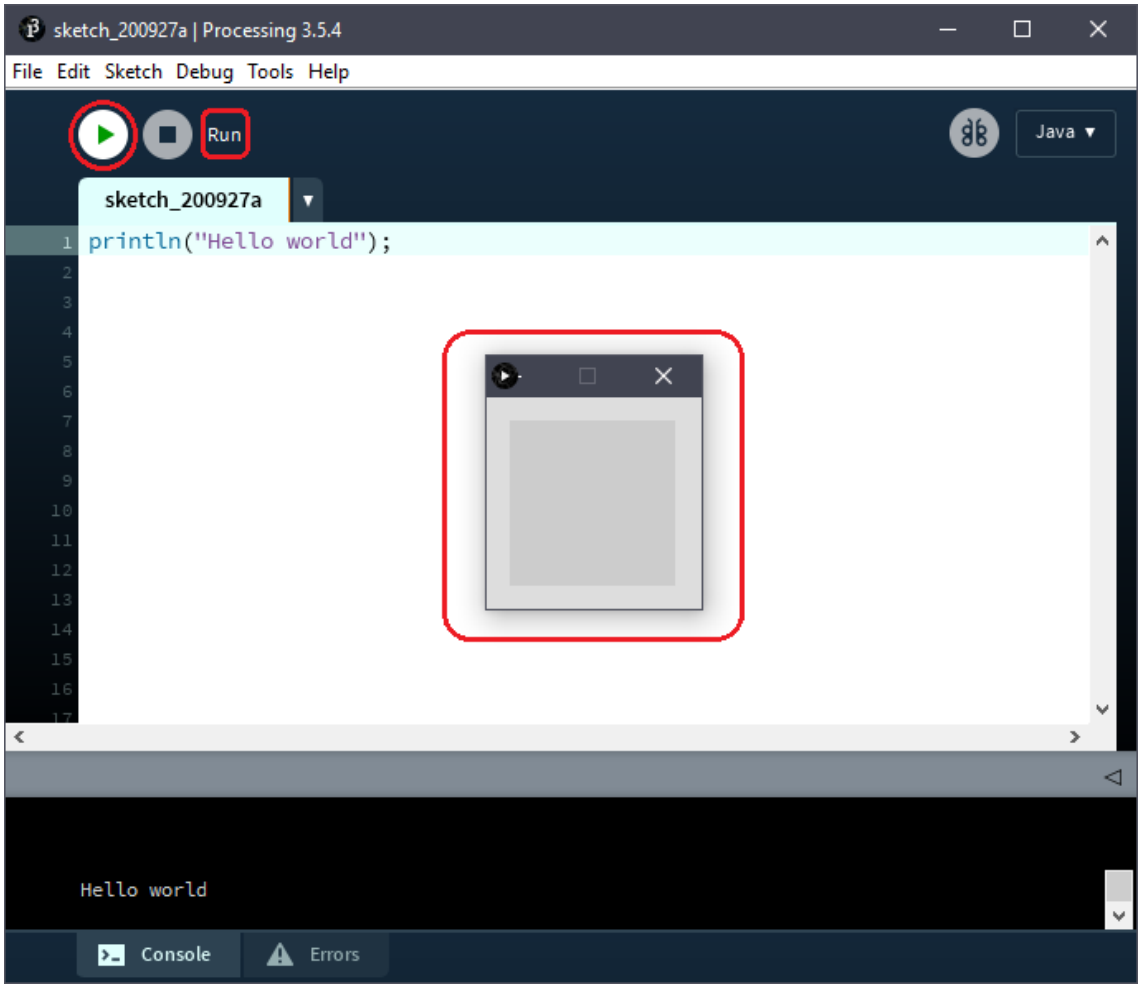
Ανοίγοντας το Processing δημιουργείται αυτόματα ένα νέο project στο οποίο μπορούμε αμέσως να πληκτρολογήσουμε εντολές σε γλώσσα Java ακόμη κι ας μην αφορούν το γραφικό περιβάλλον του Processing.

Ο επεξεργαστής κειμένου βρίσκεται στο κύριο παράθυρο διαλόγου του Processing **IDE** (Εικόνα 2) και συνοδεύεται από την κονσόλα (console) αλλά και τον τομέα διαχείρισης των σφαλμάτων που παράγονται.

Ο καμβάς βρίσκεται σε ξεχωριστό παράθυρο το οποίο εμφανίζεται κάθε φορά που αποφασίζουμε να εκτελέσουμε τον εκάστοτε κώδικα, για την εκτέλεση πατάμε το κουμπί **Run** (Εικόνα 3). Ο καμβάς εμφανίζεται ανεξάρτητα με το αν υπάρχουν εντολές κώδικα.



Εικόνα 2: Processing IDE.



Εικόνα 3: Καμβάς (Sketch)

2.2 Κατανόηση της λειτουργικότητας του καμβά (sketch):

Πριν ξεκινήσουμε τη σχεδίαση θα πρέπει πρώτα να εξοικειωθούμε με το περιβάλλον και τις ιδιομορφίες του καμβά. Ο καμβάς μας ξεκινά με ένα προκαθορισμένο (default) μέγεθος το οποίο όμως δεν μας παρέχει ιδιαίτερη άνεση χώρου (Εικόνα 3).

Υπάρχει λοιπόν η μέθοδος **size()** η οποία δέχεται δύο έως τρείς παραμέτρους, οι δύο πρώτες εκ των οποίων καθορίζουν αντίστοιχα το πλάτος και το ύψος του παραθύρου του καμβά με μονάδα μέτρησης τα pixel π.χ. η εντολή «**size(320, 240);**», δηλαδή το παράθυρο διαλόγου θα είναι 320 pixels σε πλάτος και 240 pixels σε ύψος. Η εντολή size θα πρέπει να βρίσκεται αναγκαστικά εμφωλευμένη και στη πρώτη γραμμή της μεθόδου **setup()** (Εικόνα 4)**.**

Η μέθοδος setup εκτελείται μόνο μία φορά κατά την εκκίνηση του προγράμματος και χρησιμοποιείται για την αρχικοποίηση μεταβλητών, ιδιοτήτων του περιβάλλοντος όπως το μέγεθος του καμβά ή για τη φόρτωση οπτικών μέσων όπως εικόνες.

Σε αυτό το σημείο πρέπει να λάβουμε υπόψιν μας μία ακόμη παράμετρο. Ο καμβάς έχει ως ιδιότητα τον ρυθμό ανανέωσης καρέ ανά δευτερόλεπτο (framerate), αυτή η ιδιότητα μας βοηθά στο να προσομοιώνουμε μορφές κίνησης των γραφικών στοιχείων αλλά και να κάνουμε ταυτόχρονους υπολογισμούς για οντότητες που μπορεί να εμπεριέχονται στο περιβάλλον.

Εφόσον η μέθοδος **setup** προσπελαύνεται μόνο μια φορά κατά τη διάρκεια της εκτέλεσης του προγράμματος, δημιουργείται η ανάγκη για μία διαδικασία που θα μας εμφανίζει τα γραφικά που σχεδιάζουμε σε κάθε καρέ (frame). Δηλαδή σε κάθε καρέ να έχουμε μια επαναλαμβανόμενη ανανέωση του περιβάλλοντος και των στοιχείων του. Η μέθοδος που αναλαμβάνει αυτή τη λειτουργία είναι η **draw()**.

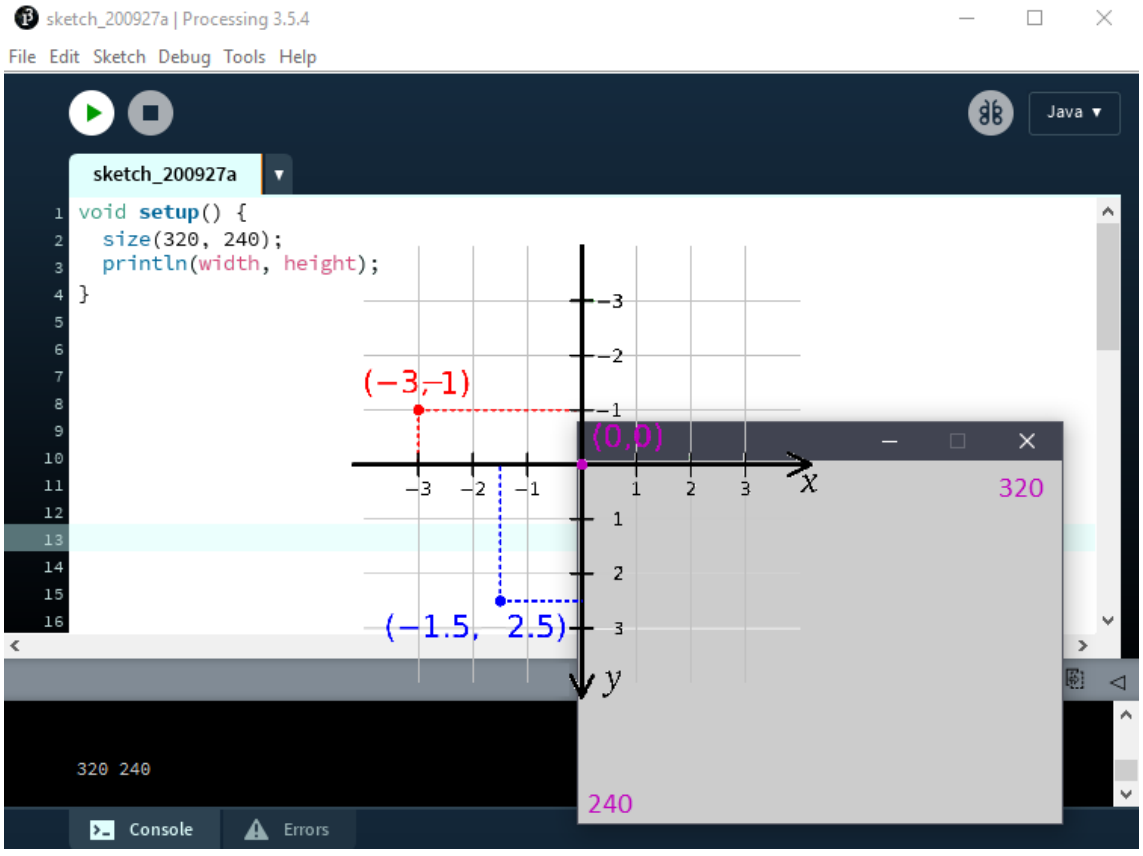
Η draw καλείται αμέσως μετά την setup και εκτελεί συνεχώς τις γραμμές κώδικα που εμπεριέχει μέχρι να τερματίσει ο χρήστης το πρόγραμμα. Η ανανέωση του κάθε καρέ γίνεται αφού εκτελεστούν όλες οι εντολές της draw.

Είναι σημαντικό να θυμόμαστε ότι έχουμε τη δυνατότητα να σχεδιάσουμε οτιδήποτε επιθυμούμε γράφοντας εντολές σχεδίασης στην setup. Έπειτα όμως καλείται η draw με αποτέλεσμα να ανανεωθεί ο καμβάς αντικαθιστώντας έτσι οτιδήποτε υπήρχε στο προηγούμενο καρέ.

Το τελευταίο βήμα προτού ξεκινήσουμε τη σχεδίαση είναι να προσδιορίσουμε το σύστημα συντεταγμένων που χρησιμοποιεί το Processing ώστε να γνωρίζουμε πως να τοποθετούμε με ακρίβεια γραφικά στοιχεία στον καμβά.

Όπως φαίνεται στην παρακάτω εικόνα (Εικόνα 4), το σύστημα συντεταγμένων που χρησιμοποιείται θα μπορούσαμε να το παρομοιάσουμε ως ένα καρτεσιανό σύστημα συντεταγμένων με άξονα τετμημένων **X** και άξονα τεταγμένων **Y** με τον Y όμως ανεστραμμένο. Άρα μιλάμε για ένα γραφικό περιβάλλον δύο διαστάσεων (**2D**) με οριζόντιο άξονα τον X και κάθετο άξονα τον Y.

Το αποτέλεσμα της τομής των δύο αξόνων δημιουργεί το σημείο της αρχής των αξόνων **(0, 0)** και βρίσκεται στο πάνω αριστερό άκρο του παραθύρου του καμβά. Ο άξονας X στο παράδειγμα (Εικόνα 4) έχει ελάχιστη τιμή το **0** ξεκινώντας από τα αριστερά και μέγιστη **320** (pixel) φτάνοντας στο τέλος του παραθύρου στα δεξιά ενώ ο άξονας Y έχει ελάχιστη τιμή το **0** ξεκινώντας από τη κορυφή του παραθύρου και μέγιστη **240** (pixel) φτάνοντας στον πάτο του παραθύρου.



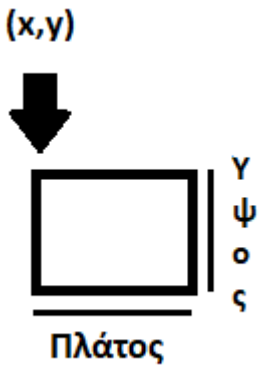
Εικόνα 4: Σύστημα συντεταγμένων.

2.3 Βασικές αρχές σχεδίασης:

Στόχος μας είναι να σχεδιάσουμε γραφικά στον καμβά μας. Άρα χρειάζεται να συντάξουμε συγκεκριμένες εντολές σχεδίασης ώστε να εμφανιστούν τα διάφορα στοιχεία μέσω της εκτέλεσης του κώδικα.

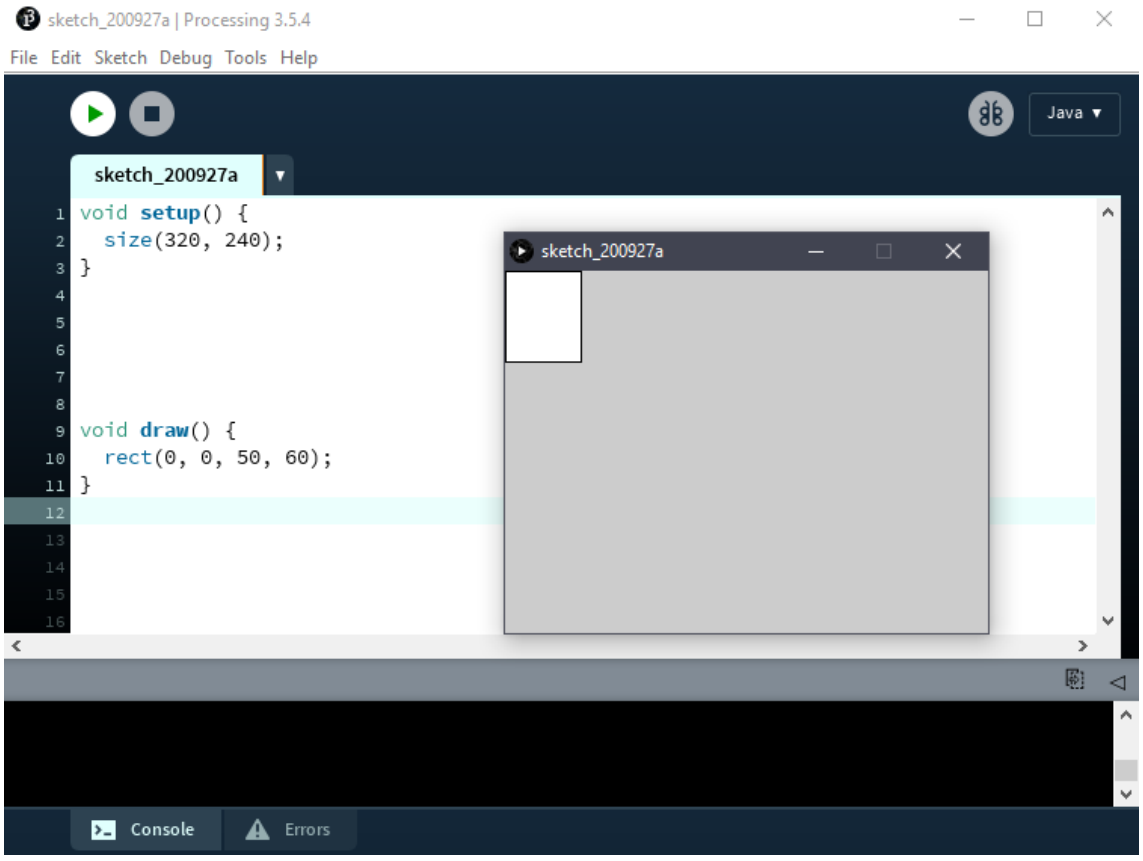
Το Processing έχει έτοιμες υλοποιημένες κλάσεις με τις οποίες μπορούμε να δημιουργήσουμε σχήματα όπως ένα ορθογώνιο παραλληλόγραμμο (**rectangle**) μία έλλειψη (**ellipse**) άλλα και πιο περίπλοκα σχέδια που μπορούμε να σχεδιάσουμε μόνοι μας με την βοήθεια συγκεκριμένων εντολών.

Η μέθοδος που υλοποιεί ένα ορθογώνιο παραλληλόγραμμο είναι η **rect()** και δέχεται τέσσερις πολύ συγκεκριμένες παραμέτρους**.** Οι δύο πρώτες παράμετροι ορίζουν τη θέση της επάνω αριστερά γωνίας στους άξονες X/Y , η τρίτη ορίζει το πλάτος και η τέταρτη ορίζει το ύψος του σχήματος (Εικόνα 5).



Εικόνα 5: Παράμετροι σχήματος.

Συγκεκριμένα η εντολή «**rect(0, 0, 50, 60);**» θα δημιουργήσει ένα ορθογώνιο παραλληλόγραμμο όπου η πάνω αριστερή γωνία του σχήματος θα σχεδιαστεί στη θέση 0 στον άξονα X, 0 στον άξονα Y με πλάτος 50 και ύψος 60, άρα η πάνω αριστερή γωνία του σχήματος θα σχεδιαστεί στην αρχή των αξόνων όπως φαίνεται στην παρακάτω εικόνα (Εικόνα 6). Το σχήμα καθώς σχεδιάζεται εκτείνεται διαγώνια προς τα δεξιά με βάση τις συντεταγμένες x/y. Παρατηρούμε ότι η εντολή rect βρίσκεται μέσα στη μέθοδο draw, αυτό σημαίνει ότι το σχήμα μας θα εμφανίζεται σε κάθε καρέ στο ίδιο σημείο στον καμβά.



Εικόνα 6: Απεικόνιση σχεδίασης στοιχείου.

ΚΕΦΑΛΑΙΟ 3: ΜΟΝΤΕΛΟΠΟΙΗΣΗ ΦΥΣΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

3.1 Μοντελοποίηση φυσικών συστημάτων:

Οποιαδήποτε στιγμή έχουμε τη δυνατότητα να σχεδιάσουμε μια ή περισσότερες οντότητες στο περιβάλλον μας, παρόλα αυτά μια οντότητα όπως ένα ορθογώνιο παραλληλόγραμμο αποτελεί μια άψυχη απεικόνιση στην οθόνη μας. Σε αυτό το κεφάλαιο θα μελετήσουμε τρόπους τέτοιους ώστε να μοντελοποιήσουμε ένα διαδραστικό περιβάλλον με οντότητες που κινούνται στο χώρο και λαμβάνουν αποφάσεις σε σχέση με αυτόν.

Η προσπάθεια της μοντελοποίησης ενός φυσικού συστήματος προσανατολίζεται στο να προσομοιώσουμε όσο το δυνατόν περισσότερο τις δυνάμεις που επιδρούν στα σώματα στον πραγματικό κόσμο σε απλουστευμένη μορφή στον κώδικά μας προσαρμόζοντάς τα στο περιβάλλον που μας αφορά.

Θα χρησιμοποιήσουμε νόμους και τύπους της φυσικής για την αναπαράσταση των διαφόρων διανυσμάτων που παράγονται από την επίδραση δυνάμεων στα αντικείμενα του περιβάλλοντος μας.

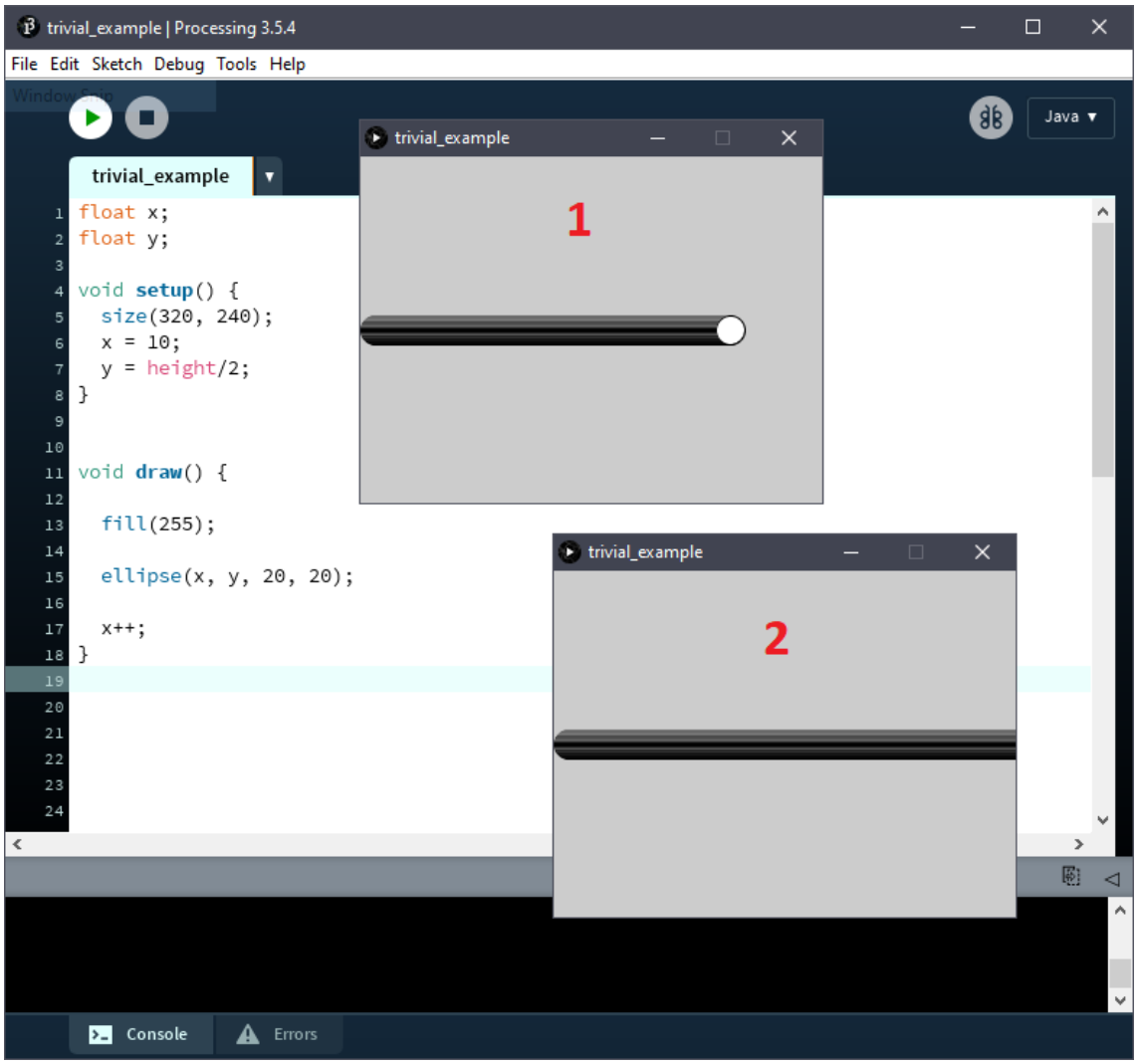
Υπάρχουν ήδη ανεπτυγμένες βιβλιοθήκες (**physics engines**) που αναλαμβάνουν την πραγματοποίηση τέτοιων σεναρίων, ωστόσο για τις ανάγκες της εργασίας θα δημιουργήσουμε την δική μας βιβλιοθήκη κατά κάποιον τρόπο.

3.2 Απλές κινήσεις οντοτήτων:

Όλες οι σχεδιαστικές οντότητες - τα αντικείμενα στο Processing λαμβάνουν ως παράμετρο τιμές θέσης στους άξονες X/Y όπως και τιμές για το μέγεθός τους.

Ξεκινώντας από τις απλές κινήσεις γνωρίζουμε ότι στον καμβά μας ένα αντικείμενο μπορεί να σχεδιαστεί στους άξονες X/Y άρα μπορούμε να μεταβάλλουμε τις τιμές x και y για να δώσουμε την αίσθηση της κίνησης του αντικειμένου στον χώρο.

Εάν επιθυμούμε να δημιουργήσουμε το σενάριο όπου ένα αντικείμενο κινείται σταθερά στον άξονα X θα πρέπει σε κάθε καρέ να επηρεάζουμε με κάποιον τρόπο τη συντεταγμένη x του εκάστοτε αντικειμένου. Παρακάτω προσομοιώσουμε μία έλλειψη η οποία κινείται οριζόντια και προς τα δεξιά μέχρι να ξεπεράσει τα χωρικά όρια του καμβά (Εικόνα 7).



Εικόνα 7: Κίνηση αντικειμένου.

Παρατηρούμε λοιπόν στην παραπάνω εικόνα ότι αρχικοποιούμε την μεταβλητή που έχουμε ονομάσει x με την τιμή **10** και την μεταβλητή y με την τιμή height/2. Αυτό σημαίνει ότι η έλλειψη θα σχεδιαστεί στο πρώτο καρέ στον οριζόντιο άξονα στην θέση **10** και στον κάθετο άξονα στο μέσο της οθόνης.

Η λέξη **height** είναι μια δεσμευμένη λέξη της γλώσσας που έχει καταχωρημένη ως σταθερά (**built-in variable**) την τιμή του ύψους του παραθύρου του καμβά. Με την ίδια λογική υπάρχει η αντίστοιχη δεσμευμένη λέξη για το πλάτος του παραθύρου με την κωδική ονομασία **width**.

Έπειτα παρατηρούμε μια ακόμη νέα εντολή, την **fill().** Η fill δέχεται τιμές από **0** έως **255**, χρησιμεύει στο να χρωματίζει τα αντικείμενα που σχεδιάζονται στον καμβά και βρίσκεται πάντοτε πριν την εντολή σχεδίασης. Η τιμή **0** δίνει ως αποτέλεσμα το μαύρο χρώμα, η τιμή **255** το λευκό χρώμα ενώ το εύρος ανάμεσά τους δίνει τις αποχρώσεις του γκρι.

Υπάρχει βέβαια και η δυνατότητα της fill να δεχτεί έως και τέσσερις παραμέτρους με τον ρόλο των τριών πρώτων να ορίζουν την δυνατότητα απεικόνισης **RGB** χρωματικών συνδυασμών αντίστοιχα, ενώ η τέταρτη προσδίδει την ιδιότητα της διαφάνειας του σχήματος (**transparency**).

Τέλος η εντολή επαύξησης της τιμής x «**x++;**» δίνει σε κάθε καρέ μία νέα θέση σχεδίασης της έλλειψης, δηλαδή ένα pixel στον άξονα X σε κάθε επανάληψη ενώ η τιμή της y παραμένει σταθερή.

Στο παραπάνω παράδειγμα επιτρέψαμε να σχηματιστεί αυτό το μονοπάτι για να αποδείξουμε στις εικόνες ότι το αντικείμενο κινούταν στον χώρο. Το μονοπάτι αυτό δημιουργήθηκε διότι σε κάθε καρέ δεν σβηνόταν ο σχεδιασμός της έλλειψης στο αμέσως προηγούμενο σημείο στον άξονα X με αποτέλεσμα να σχεδιάζονται δεκάδες ελλείψεις στην οθόνη. Για να εξαλείψουμε αυτό το μονοπάτι και να δημιουργήσουμε μια πιο ευχάριστη και αληθοφανή εικόνα κίνησης θα πρέπει να εισάγουμε μια νέα εντολή στο πρόγραμμά μας. Η εντολή αυτή είναι η **background()** και δέχεται τις ίδιες παραμέτρους με την fill μόνο που επιδρά στο χρώμα του καμβά και όχι στων αντικειμένων.

Η εντολή background θα πρέπει να είναι η πρώτη εντολή της μεθόδου draw ώστε κάθε φορά που θα ολοκληρώνεται ένας σχεδιασμός να τον διαδέχεται ο επόμενος σε κενό καμβά δημιουργώντας έτσι την ψευδαίσθηση ενός κινουμένου σώματος που δεν αφήνει ίχνη.

Κάποια στιγμή η τιμή του x ξεπερνά την τιμή του πλάτους του παραθύρου με αποτέλεσμα η έλλειψη να βγαίνει εκτός ορίων και να χάνεται από το οπτικό πεδίο, παρόλα αυτά η έλλειψη συνεχίζει να κινείται στον χώρο χωρίς να την βλέπουμε εφόσον δεν έχει τερματιστεί το πρόγραμμα. Σε αυτήν την περίπτωση θα πρέπει να εισάγουμε εντολές που θα εφαρμόζουν κάποια όρια στο αντικείμενό μας (Εικόνα 8).

Αρχικά στην παρακάτω εικόνα παρατηρούμε ότι υπάρχει ακόμη το μονοπάτι της διαδρομής που ακολούθησε το αντικείμενό μας και αυτό συμβαίνει γιατί η εντολή background υπάρχει σε μορφή σχολίου με τη σήμανση των δύο καθέτων (//) οπότε είναι ανενεργή για τους λόγους που εξηγήσαμε παραπάνω.

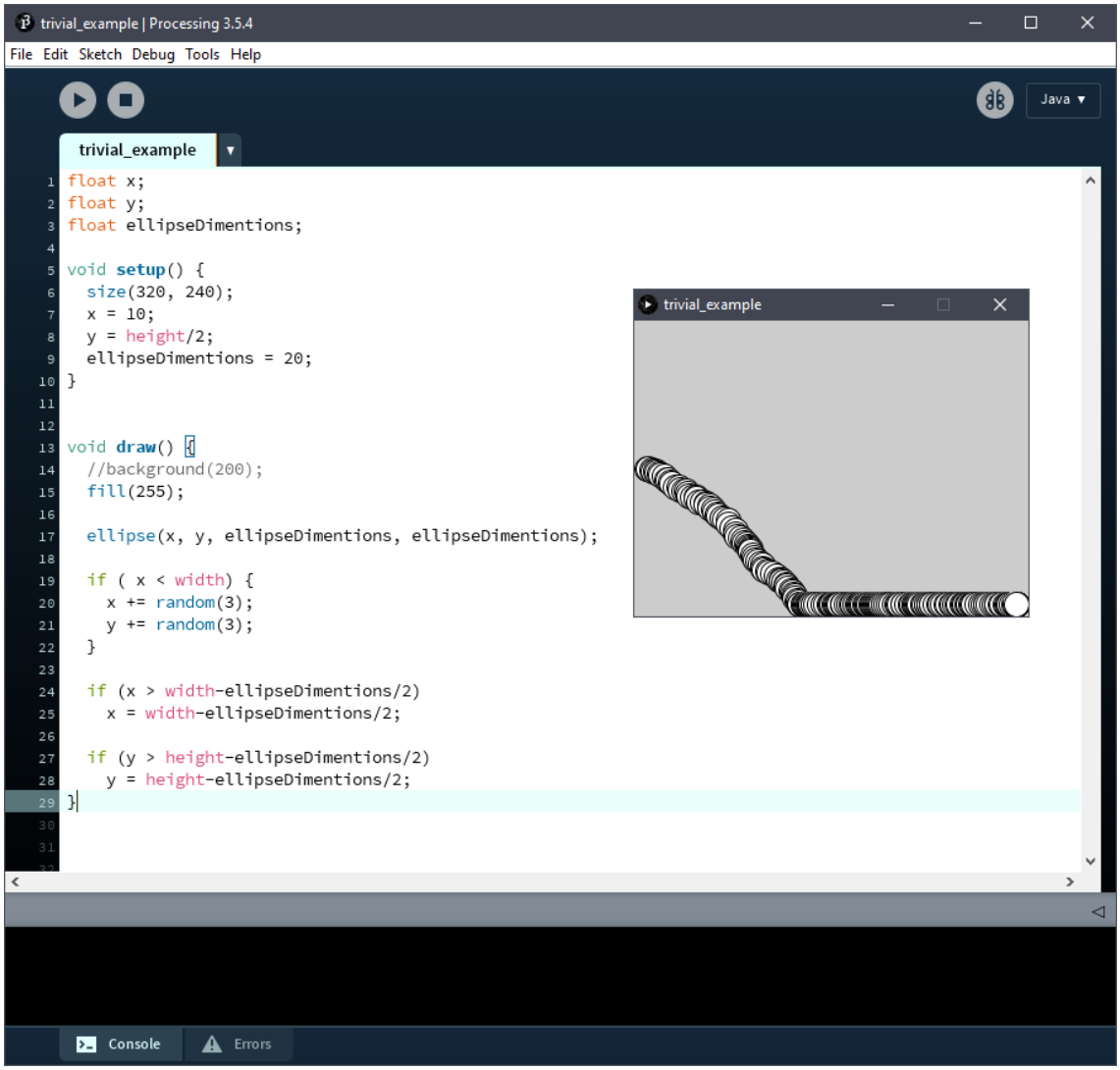
Παρατηρούμε επιπλέον πως οι διαστάσεις του αντικειμένου πλάτος και ύψος της έλλειψης αντικαταστάθηκαν από μία μεταβλητή που ονομάσαμε **ellipseDimentions** έτσι δημιουργούμε μία πιο δυναμική έκδοση του προγράμματος.

Στη γραμμή **19** του κώδικα της εικόνας έχουμε μια έκφραση **if** η οποία εφόσον είναι αληθής σε κάθε επανάληψη της draw οι τιμές x και y θα λαμβάνουν μια τυχαία τιμή η οποία θα προστίθενται στο σύνολο της κάθε μεταβλητής.

Η εντολή **random()** δέχεται από μία έως δύο παραμέτρους, στην περίπτωση της μίας δίνει αποτελέσματα από **0** έως και έναν αριθμό πριν από αυτόν που περικλείεται, στην δική μας περίπτωση από **0** έως και **2**.

Σε αυτό το παράδειγμα οι τιμές X/Y μεταβάλλονται προκαλώντας την μεταβολή της θέσης του αντικειμένου μας στους αντίστοιχους άξονες δημιουργώντας έτσι ένα διάνυσμα το οποίο εκφράζει τη μεταβολή της μετατόπισης αντικειμένου στον χώρο. Άρα όπως προκύπτει η επαύξηση των μεταβλητών x και y σε κάθε καρέ με συγκεκριμένο ρυθμό περιγράφει το διάνυσμα της ταχύτητας του αντικειμένου.

Οι δύο επόμενες εκφράσεις if έχουν σκοπό να περιορίσουν το αντικείμενο από το να φύγει από την οθόνη εφόσον φτάσει στα όρια του δεξιού και του κάτω φράγματος του καμβά, φυσικά αυτό θα μπορούσε να επεκταθεί και για όλες τις πλευρές του παραθύρου.



Εικόνα 8: Οριοθέτηση αντικειμένου.

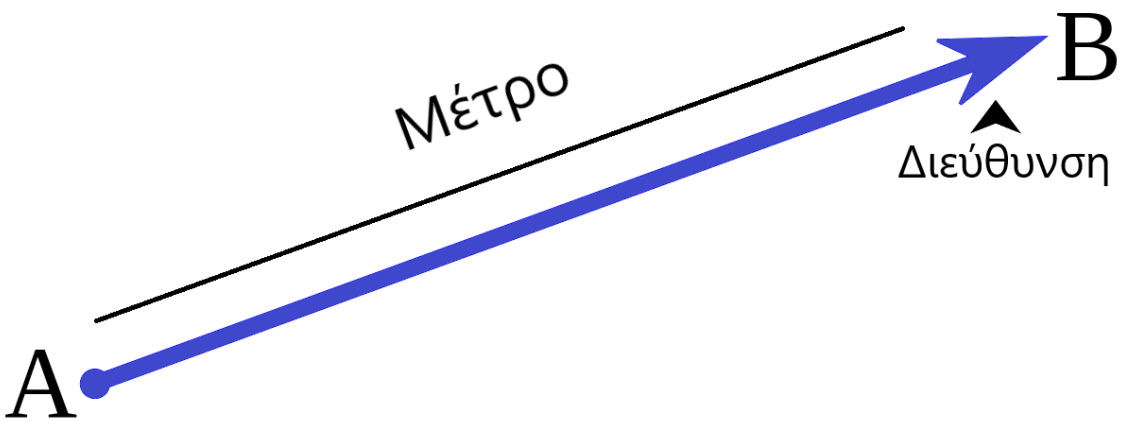
3.3 Διανύσματα:

Στη προηγουμένη υποενότητα αναφερθήκαμε στη μεταβολή της θέσης του αντικειμένου ως ένα διάνυσμα. Χρησιμοποιώντας την ορολογία του διανύσματος θα αναφερόμαστε στο Ευκλείδειο διάνυσμα.

Ένα διάνυσμα μπορεί να διατυπωθεί ως ένα γεωμετρικό αντικείμενο που το χαρακτηρίζουν δύο ιδιότητες. Οι ιδιότητες αυτές είναι το μέτρο (magnitude) και η διεύθυνση (direction).

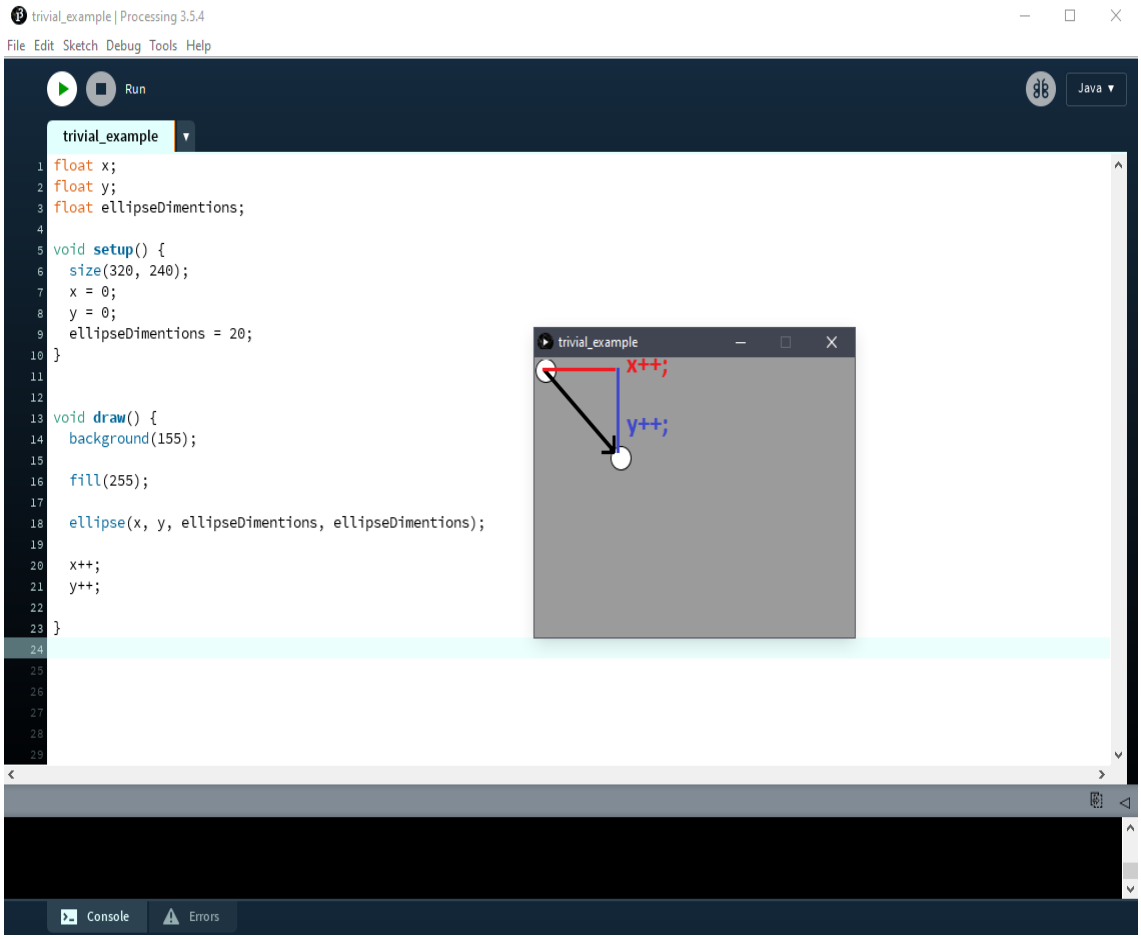
Μεταξύ διανυσμάτων υφίστανται οι μαθηματικές πράξεις του αθροίσματος και της αφαίρεσης σύμφωνα με τους κανόνες της διανυσματικής άλγεβρας πράγμα που αποδεικνύεται πολύ χρήσιμο σε περιπτώσεις που μία οντότητα καλείται να διαχειριστεί πολλές δυνάμεις.

Ένα διάνυσμα μπορεί να αναπαρασταθεί με ένα βέλος. Το μέτρο του διανύσματος είναι η απόσταση μεταξύ δύο σημείων και η διεύθυνση αναπαριστά τη κατεύθυνση προς την οποία μεταφέρεται (Εικόνα 9). Δηλαδή ένα διάνυσμα είναι μια οδηγία που προσδιορίζει το πού και πόσο γρήγορα θα μεταφερθεί. Τα διανύσματα μπορούν να εκφράσουν διάφορα είδη διανυσματικών μεγεθών όπως η ταχύτητα και η μετατόπιση.



Εικόνα 9: Απεικόνιση διανύσματος.

Η θέση του αντικειμένου είναι επίσης ένα διάνυσμα που μας δίνει ένα σημείο (x/y) της τελικής θέσης σχεδίασης σε σχέση με την αρχική ενώ το διάνυσμα της ταχύτητας εκφράζει το πώς θα κινηθεί από ένα σημείο σε ένα άλλο (Εικόνα 10).



Εικόνα 10: Διάνυσμα θέσης.

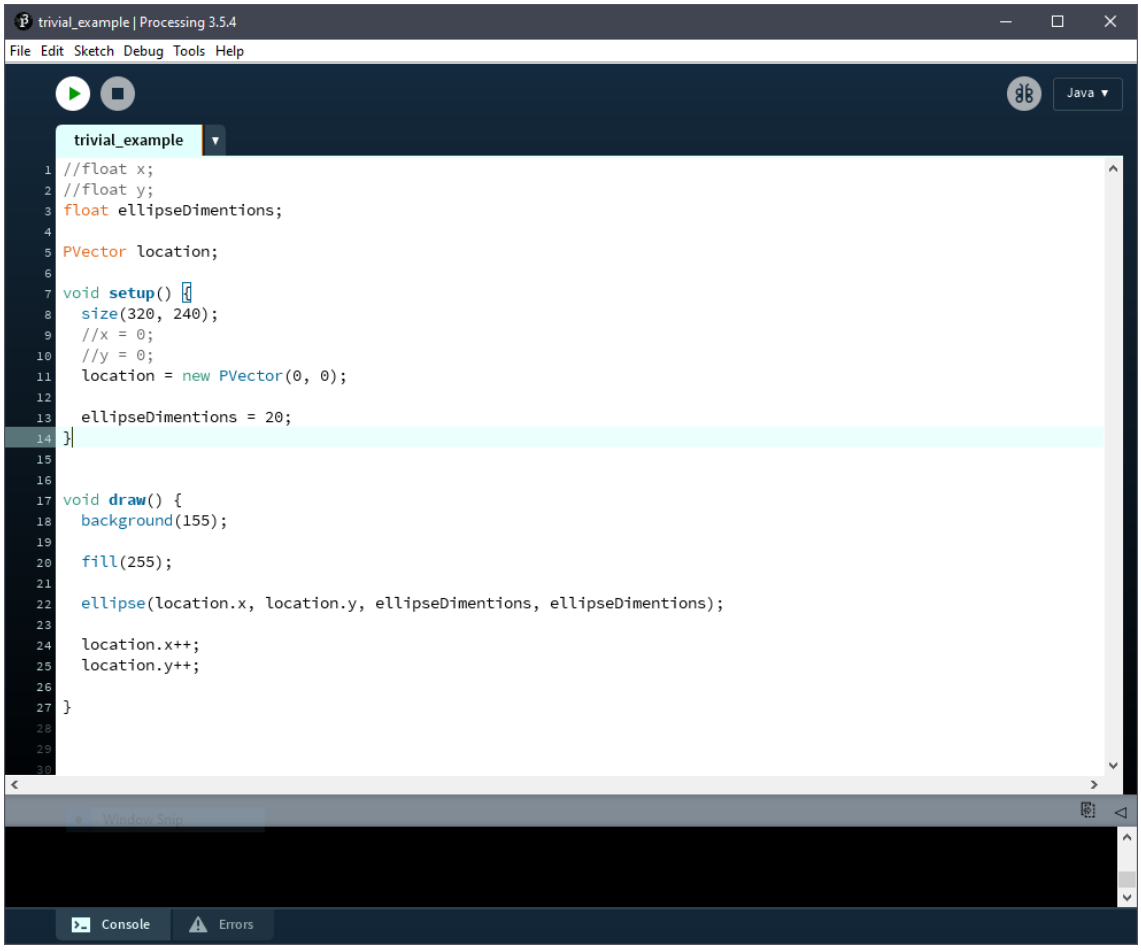
3.4 Εφαρμογή διανυσμάτων:

Η ύπαρξη μίας οντότητας στον χώρο αλλά και η κίνηση της δίνεται πάντοτε σε συνάρτηση με τις συντεταγμένες σχεδίασης x και y της οντότητας σε σχέση με αυτές του καμβά.

Πλέον οι οντότητες μας θα έχουν ιδιότητες σε μορφή μεταβλητών που θα περιγράφουν τα διανύσματα θέσης (**location**), ταχύτητας (**velocity**) και επιτάχυνσης (**acceleration**) που διαθέτουν.

Τα παραπάνω μεγέθη είναι αλληλεξαρτώμενα, για παράδειγμα η μεταβολή (ανά καρέ/χρονική στιγμή) της μετατόπισης στον χώρο εκφράζει το διάνυσμα της ταχύτητας καθώς επίσης η μεταβολή της ταχύτητας στο χρόνο εκφράζει το διάνυσμα της επιτάχυνσης.

Ιδανικά μας ενδιαφέρει να αντικαταστήσουμε τις μεταβλητές x και y σε μια γενικότερη μεταβλητή που θα τις ομαδοποιεί. Το Processing περιλαμβάνει μια κλάση η οποία μας επιτρέπει να υλοποιήσουμε τις παραπάνω παραδοχές. Η κλάση αυτή ονομάζεται **PVector** και περιγράφει ένα οποιοδήποτε διάνυσμα αποθηκεύοντας μεταβλητές συντεταγμένων x/y.



Εικόνα 11: Κλάση PVector.

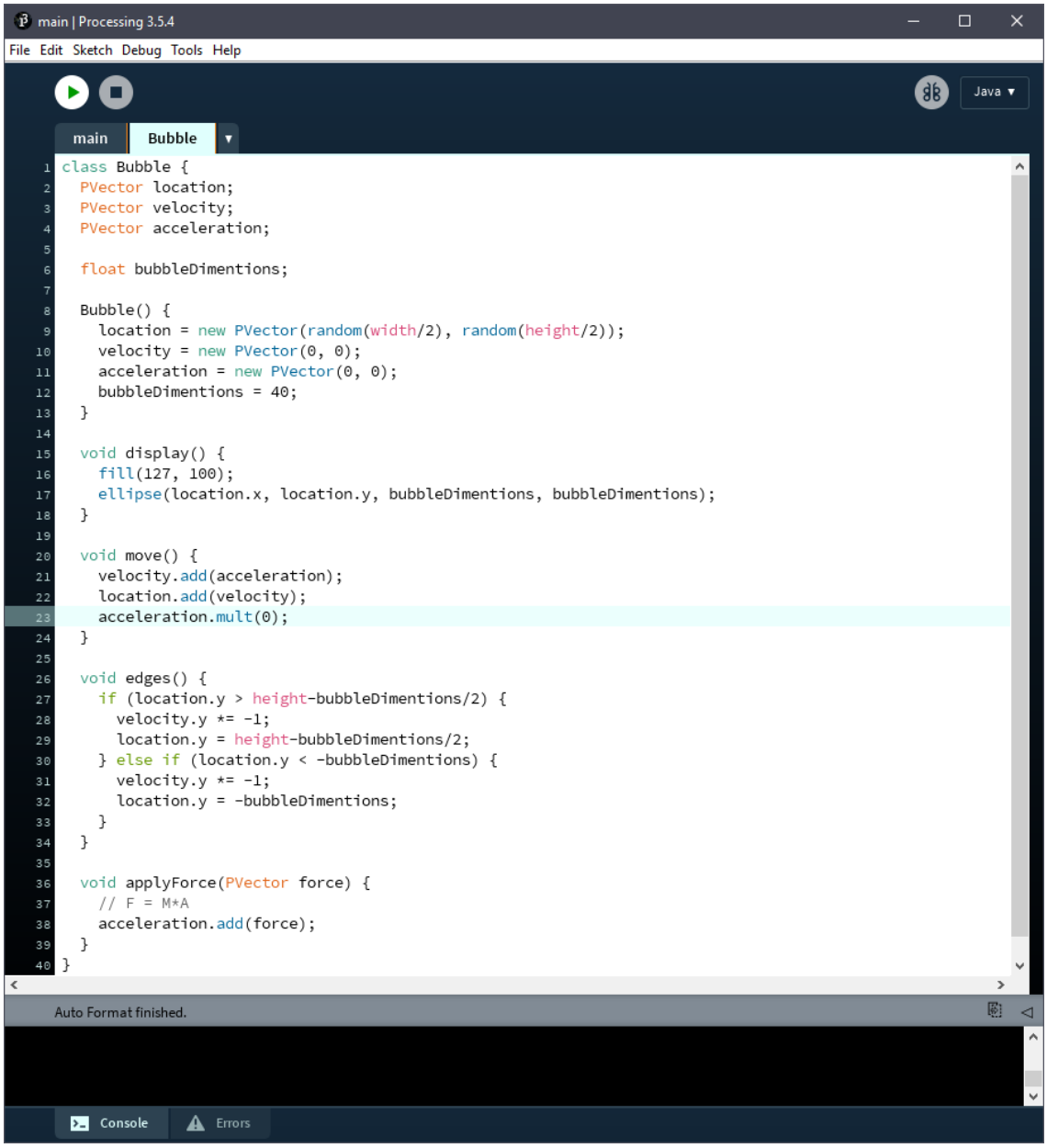
Στο παραπάνω παράδειγμα (Εικόνα 11) διατηρήσαμε την λειτουργικότητα του παραδείγματος της εικόνας 10 μόνο που αυτή τη φορά αντικαταστήσαμε τις μεταβλητές x/y με ένα αντικείμενο PVector που ονομάσαμε **location**.

Για να προσπελάσουμε τις τιμές x και y αντίστοιχα θα πρέπει να αναφερθούμε πρώτα το όνομα του αντικειμένου ακολουθούμενο από μια τελεία (**.**) (dot operator) και στη συνέχεια στην μεταβλητή που θέλουμε να αντλήσουμε π.χ **location.x** ή **location.y**.

3.5 Εφαρμογή λειτουργικότητας σε αντικείμενα κλάσεων:

Εφόσον μάθαμε να υλοποιούμε διανύσματα επόμενο βήμα είναι να δημιουργήσουμε αληθοφανείς αλληλεπιδράσεις μεταξύ άλλων διανυσμάτων.

Όπως εξηγήσαμε στην προηγούμενη υποενότητα τα διανύσματα θέσης, ταχύτητας και επιτάχυνσης είναι αλληλεξαρτώμενα, άρα εάν ισχύει ότι η μεταβολή της μετατόπισης στον χώρο εκφράζει το διάνυσμα της ταχύτητας τότε χρειαζόμαστε έναν τρόπο να κάνουμε πράξεις μεταξύ των διανυσμάτων. Η κλάση PVector έχει υλοποιημένες μεθόδους που αναλαμβάνουν τις διάφορες μαθηματικές πράξεις που χρησιμοποιούνται συνήθως στα διανύσματα.



Εικόνα 12: Κλάση Bubble.

Στο παράδειγμα της εικόνας 12 παρατηρούμε ότι έχουμε δημιουργήσει μια κλάση με το όνομα **Bubble** που περιγράφει μια οντότητα «φούσκα». Ξεκινώντας από τη δήλωση των μεταβλητών φαίνονται τα ιδιαίτερα χαρακτηριστικά της οντότητας, τα διανύσματα και οι διαστάσεις του αντικειμένου.

Έπειτα με τη δημιουργία του αντικειμένου αρχικοποιούνται όλες οι μεταβλητές, στη συνέχεια έχουμε υλοποιήσει μεθόδους η κάθε μία από τις οποίες προσδίδει μία λειτουργικότητα στην οντότητά μας. Αρχικά η μέθοδος **display()** που μας επιτρέπει να εμφανίζουμε το αντικείμενο στην οθόνη μας.

Η επόμενη κατά σειρά μέθοδος είναι η **move()** που αναλαμβάνει την κίνηση του αντικειμένου.

Η πρώτη εντολή είναι η «**velocity.add(acceleration);»,** η μέθοδος **add()** προσθέτει τα στοιχεία (μέγεθος, διεύθυνση) ενός διανύσματος σε ένα άλλο. Στη συγκεκριμένη περίπτωση οι τιμές x και y της επιτάχυνσης (acceleration) προστίθενται στις αντίστοιχες τιμές της ταχύτητας (velocity). Οι τιμές και των δύο διανυσμάτων αρχικά είναι μηδέν, στη συνέχεια θα δούμε πως ένα ερέθισμα θα δώσει κίνηση στο σώμα.

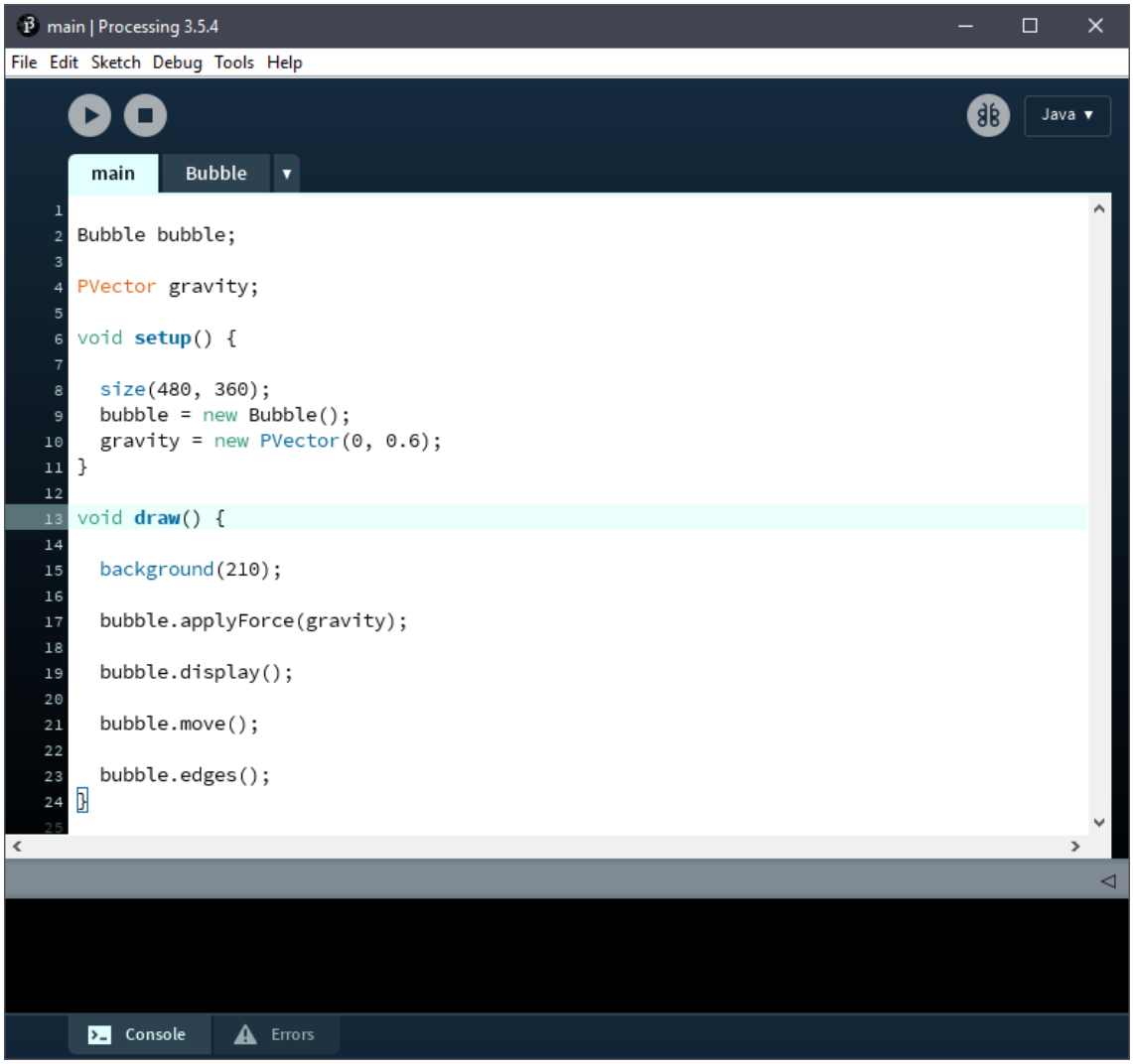
Με την ίδια λογική όπως παραπάνω εκτελείται η «**location.add(velocity);**», η ταχύτητα αλλάζει τις τιμές των συντεταγμένων της θέσης του αντικειμένου με αποτέλεσμα την κίνηση του στον χώρο.

Η τελευταία εντολή είναι η «**acceleration.mult(0);»,** η μέθοδος **mult()** πολλαπλασιάζει τα στοιχεία ενός διανύσματος με ένα βαθμωτό μέγεθος. Στην περίπτωσή μας είναι το μηδέν και αυτό για να μηδενίζουμε την επιτάχυνση σε κάθε καρέ. Η επιτάχυνση συσσωρεύεται πολύ γρήγορα οπότε για να αποφύγουμε την ανεξέλεγκτη επιτάχυνση του αντικειμένου μας χρησιμοποιούμε αυτή τη μέθοδο αλλιώς θα έπρεπε να γράψουμε «**acceleration.x = 0;»** και «**acceleration.y = 0;»**

Η μέθοδος **edges()** αναλαμβάνει την οριοθέτηση της κίνησης της φούσκας στον κάθετο άξονα ώστε να μην υπερβαίνει τα όρια του παραθύρου του καμβά. Κάθε στιγμή ελέγχεται εάν το αντικείμενό μας φτάνει στις άκρες, εάν αληθεύει αυτή η περίπτωση τότε η μέθοδός εφαρμόζει αρνητικές τιμές στη ταχύτητα του αντικειμένου στον άξονα Υ με αποτέλεσμα να αλλάξει πορεία παραμένοντας στα όρια στον καμβά.

Τέλος η μέθοδος **applyForce()** λαμβάνει ένα ερέθισμα σε μορφή διανύσματος και το προσθέτει στην επιτάχυνση της φούσκας. Η συγκεκριμένη μέθοδος είναι εμπνευσμένη από τον δεύτερο νόμο του Νεύτωνα «**F = M*A**» όπου **F** μια δύναμη που επιδρά σε ένα σώμα, **M** η μάζα του σώματος και **A** η επιτάχυνση του σώματος. Στις προσομοιώσεις μας η μάζα για όλα τα αντικείμενα θα είναι ίδια (μια σταθερά) άρα και αμελητέα εφόσον θα έχουν όλα το ίδιο μέγεθος, οπότε μια οποιαδήποτε δύναμη προκαλεί επιτάχυνση στο αντικείμενό μας «**F = A**».

Επομένως με την υλοποίηση των μεθόδων applyForce και move καταφέραμε να δημιουργήσουμε τη δική μας μέθοδο διαχείρισης δυνάμεων (**physics engine)**. Το οπτικό αποτέλεσμα που πετυχαίνουμε σε αυτό το παράδειγμα είναι ένα αντικείμενο που αναπηδά στον Υ άξονα.



Εικόνα 13: Αντικειμενοστραφής προσέγγιση.

Έχοντας προσδιορίσει μια κλάση οντότητας και έχοντας αναλύσει τις λειτουργείες που την διέπουν μπορούμε πλέον στο κύριο πρόγραμμα να δημιουργήσουμε αντικείμενα αυτής της κλάσης και να εφαρμόσουμε δυνάμεις – διανύσματα που θα επιδρούν σε αυτό (Εικόνα 13).

Παρατηρούμε λοιπόν στο παραπάνω παράδειγμα πως δημιουργούμε ένα αντικείμενο τύπου **Bubble** με όνομα **bubble** καθώς και ένα αντικείμενο διανύσματος με όνομα **gravity** για το οποίο θα αναφερόμαστε ως το διάνυσμα της βαρύτητας στο περιβάλλον μας (καμβά).

Η αρχικοποίηση των αντικειμένων γίνεται στην μέθοδο setup() όπου στο διάνυσμα gravity έχουν δοθεί τιμές για x ίσων με **0** δηλαδή καμία επίδραση στον άξονα X ενώ η τιμή **y** ισούται με **0.6** δηλαδή μια μικρή επίδραση στον άξονα Y.

Παρακάτω στην μέθοδο draw() καλούμε τις μεθόδους της κλάσης Bubble επί του αντικειμένου bubble με σκοπό να χρησιμοποιήσουμε τη λειτουργίες της κλάσης στο αντικείμενό μας.

Η εντολή «**bubble.applyForce(gravity);**» στέλνει το ερέθισμα του διανύσματος gravity στη μέθοδο applyForce η οποία με τη σειρά της προσθέτει τις τιμές της στην επιτάχυνση του αντικειμένου. Έπειτα καλούνται οι μέθοδοι που αναλύσαμε νωρίτερα δίνοντας τις βασικές λειτουργίες σχεδίασης και κίνησης στον χώρο.

Παρατηρούμε ότι με το να μεταφέρουμε τον κώδικα της λειτουργικότητας του αντικειμένου σε μία κλάση καταφέραμε να προσδώσουμε μια δυναμική αντικειμενοστραφή διάσταση στον κώδικά μας.

Αυτό μας βοηθά ακόμη περισσότερο να αναγνωρίσουμε τα όσα διαδραματίζονται με μία ματιά στην καρτέλα του κυρίου προγράμματος του Processing.

ΚΕΦΑΛΑΙΟ 4: ΓΕΝΕΤΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ

4.1 Ο ορισμός του γενετικού αλγορίθμου:

Χρησιμοποιώντας τον όρο «**Γενετικός Αλγόριθμος**» αναφερόμαστε σε έναν συγκεκριμένο αλγόριθμο που εφαρμόζεται με συγκεκριμένο τρόπο για την εύρεση βέλτιστων λύσεων συγκεκριμένων ειδών προβλημάτων για τα οποία δεν έχουμε τρόπο υπολογισμού μιας λύσης.

Τέτοιες περιπτώσεις προβλημάτων είναι τα συστήματα εκείνα που απαιτούν την διαχείριση πολλών παραμέτρων για τις οποίες δεν υπάρχει αναλυτική μέθοδος υπολογισμού που να καθορίζει τον βέλτιστο συνδυασμό τιμών ώστε το σύστημα να συμπεριφέρεται με τον επιθυμητό τρόπο.

Οι γενετικοί αλγόριθμοι είναι εμπνευσμένοι από τη βιολογία και τις βασικές αρχές της εξελικτικής θεωρίας του Δαρβίνου. Η λεγόμενη θεωρία του Δαρβίνου η αλλιώς Δαρβινισμός (Darwinism) αναφέρεται σε όλα τα είδη των οργανισμών που προκύπτουν στη φύση τα οποία αναπτύσσονται μέσω της φυσικής επιλογής δηλαδή των κληρονομηθέντων χαρακτηριστικών των προγόνων τους. Η κάθε οντότητα έχει στα γονίδιά της τις δικές της ιδιαίτερες γενετικές παραλλαγές οι οποίες ενδέχεται να αυξήσουν ή να μειώσουν την ικανότητα της να ανταγωνιστεί την γενιά της, να επιβιώσει στο περιβάλλον της και τις πιθανότητες της να αναπαραχθεί σε αυτό. Η θεωρία αυτή χρησιμοποιεί την ιδέα της εξέλιξης μέσω της εφαρμογής των βιολογικών μηχανισμών της φυσικής επιλογής (**natural selection**), της διασταύρωσης (**crossover**) και της γενετικής μετάλλαξης (**mutation**).

4.2 Ανάλυση των βιολογικών μηχανισμών:

4.2.1 Φυσική επιλογή:

Ξεκινώντας από την φυσική επιλογή βρισκόμαστε στο πρώτο στάδιο του γενετικού αλγορίθμου στο οποίο χρησιμοποιείται ένας πληθυσμός πιθανών λύσεων όπου κάθε δείγμα του πληθυσμού έχει ένα γονιδίωμα (**genome**) που κωδικοποιεί μια λύση, αυτό συνήθως συναντάται ως ένας συνδυασμός οδηγιών για την κάθε οντότητα.

Αργότερα επιλέγονται τα γονιδιώματα των οντοτήτων με την καλύτερη απόδοση (όσον αφορά την εύρεση της βέλτιστης λύσης) για αναπαραγωγή χρησιμοποιώντας τον μηχανισμό της διασταύρωσης.

Για να υπάρξει φυσική επιλογή όπως συμβαίνει στη φύση θα πρέπει να τεθούν οι ακόλουθες αρχές σε εφαρμογή.

Πρώτον η κληρονομικότητα, χρειαζόμαστε μια διαδικασία κατά την οποία οι απόγονοι της προηγουμένης γενιάς θα λαμβάνουν τις ιδιότητες των γονιών τους. Σε αυτό το σημείο θα πρέπει να αξιολογήσουμε την απόδοση της κάθε οντότητας με τα συγκεκριμένα χαρακτηρίστηκα. Εάν για παράδειγμα οι οντότητες έζησαν αρκετά ώστε να αναπαραχθούν ή ώστε να φτάσουν στον στόχο τους τότε τα γονίδιά τους μεταβιβάζονται στα παιδία τους ως νέοι κάτοχοι αυτών στην επόμενη γενιά.

Οπότε χρειαζόμαστε έναν μηχανισμό ή ακόμη καλύτερα μία συνάρτηση αξιολόγησης που θα εξετάζει κατά πόσο ορισμένα μέλη ενός πληθυσμού θα έχουν την ευκαιρία να γίνουν γονείς και να μεταβιβάζουν τις γενετικές τους πληροφορίες στις επόμενες γενιές. Άρα κατά κάποια έννοια θα ελέγχονται οι επιδόσεις των οντοτήτων και θα λαμβάνουν μία αντιπροσωπευτική τιμή .

Στους γενετικούς αλγορίθμους αυτή η συνάρτηση ονομάζεται συνάρτηση αξιολόγησης φυσικής κατάστασης (**fitness function**). Η fitness function είναι μια μαθηματική συνάρτηση την οποία χρησιμοποιούμε για να αξιολογήσουμε πόσο καλή είναι μια λύση. Στη συνέχεια θα δούμε πώς μια λύση με υψηλότερη φυσική κατάσταση είναι πιο πιθανό να επιλεγεί για αναπαραγωγή από ότι μια με χαμηλότερη.

Οι οντότητες που παράγουμε αναπαριστούν μία λύση στο πρόβλημά μας οπότε πρέπει να είμαστε σε θέση να αξιολογήσουμε αριθμητικά οποιαδήποτε πιθανή λύση. Η συνάρτηση αυτή θα παράγει μια αξιολόγηση με μορφή αριθμητικής βαθμολογίας περιγράφοντας τις επιδόσεις της εκάστοτε οντότητας σε έναν πληθυσμό.

Ιδανικά θέλουμε να κανονικοποιήσουμε την τιμή επίδοσης (**fitness**) της κάθε οντότητας ώστε να δουλεύουμε με τιμές από **0** έως **1** όπου αντίστοιχα θα σημαίνει χειρότερη ή καλύτερη επίδοση. Η κανονικοποίηση (**normalization**) πετυχαίνεται με τη διαίρεση της επίδοσης κάθε ατόμου με το άθροισμα των τιμών της επίδοσης όλου του πληθυσμού. Έτσι διαμορφώνουμε και τις πιθανότητες της κάθε οντότητας να επιλεγεί για αναπαραγωγή των γονιδίων του στις επόμενες γενιές.

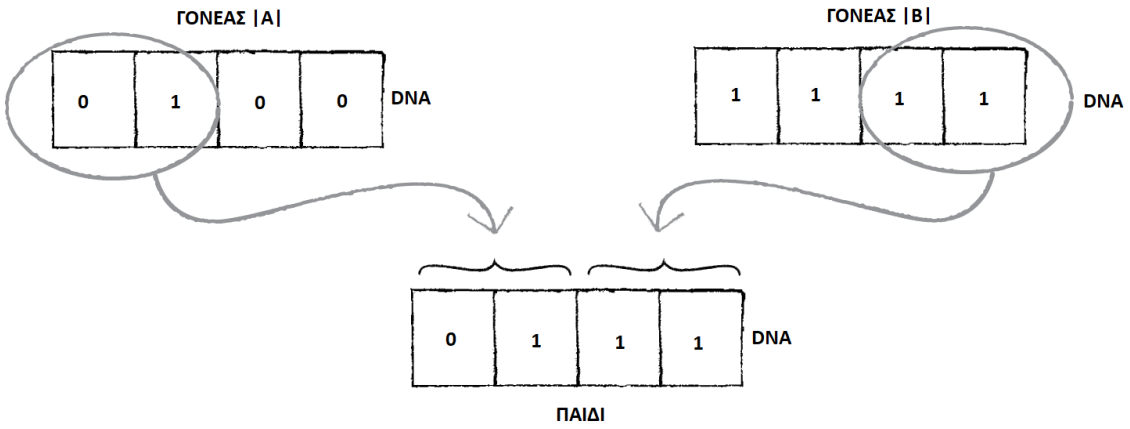
Δεύτερον η ποικιλομορφία, θα πρέπει να υπάρχει μια ποικιλία χαρακτηριστικών μεταξύ των οντοτήτων που υπάρχουν σε έναν πληθυσμό. Για παράδειγμα εάν όλες οι οντότητες ενός πληθυσμού μίας γενιάς είχαν τα ίδια ακριβώς χαρακτηριστικά όπως μέγεθος ή χρώμα τότε πάντοτε η κάθε επόμενη γενιά θα είναι ίδια με τη προγενέστερη, νέοι συνδυασμοί χαρακτηριστικών δεν μπορούν ποτέ να εμφανιστούν και ωστόσο ούτε να υπάρξει εξέλιξη.

Το ερώτημα είναι πώς μπορούμε να προσομοιώσουμε την ποικιλομορφία σε μια γενιά οντοτήτων. Για παράδειγμα αν επιθυμούσαμε οι οντότητες που έχουμε δημιουργήσει σε ένα περιβάλλον να πλοηγηθούν μέσα από ένα στενό δρομάκι τότε θα έπρεπε η κάθε οντότητα να έχει ένα αποδεκτό μέγεθος (διάπλαση) ώστε να μπορεί να το διασχίσει. Εάν είχαμε στη διάθεσή μας τρείς μόνο οντότητες με μη αποδεκτό μέγεθος τότε δεν έχουμε αρκετή ποικιλία λύσεων για να εξελιχθεί η βέλτιστη.

Ωστόσο, εάν είχαμε έναν πληθυσμό χιλιάδων οντοτήτων όλες δημιουργημένες με τυχαία μεγέθη, τότε θα είχαμε περισσότερες πιθανότητες ώστε τουλάχιστον ένα μέλος του πληθυσμού να έχει αποδεκτές τιμές. Άρα αντί να εκχωρούμε τις ιδιότητες των αντικειμένων μοντελοποιόντας πολύπλοκους αλγορίθμους για τον υπολογισμό αυτών, μπορούμε να στηριχτούμε στη διαδικασία της εξέλιξης που συναντάμε στη φύση ώστε να αποφασίζει κάθε φορά για εμάς. Ξεκινάμε λοιπόν τη πρώτη γενιά με τυχαίες τιμές για τις ιδιότητές τους και αφήνουμε τις πιθανές λύσεις να εξελιχθούν από αυτές.

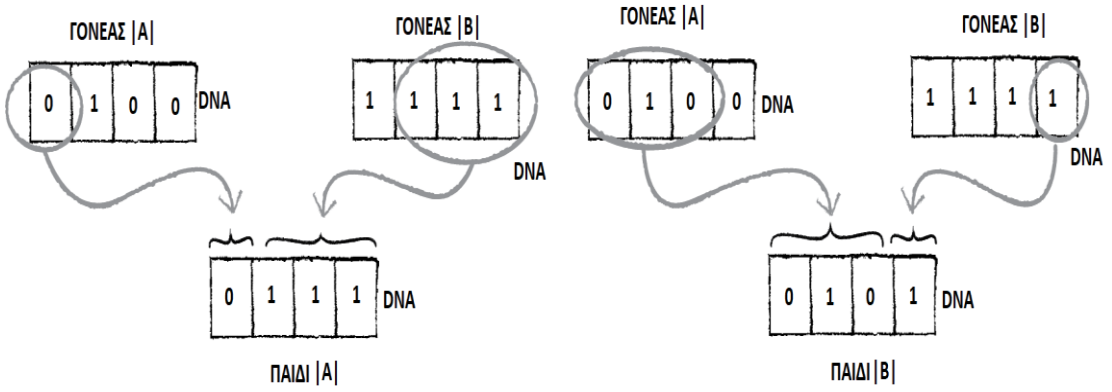
4.2.2 Διασταύρωση:

Κατά τη δημιουργία ενός νέου πληθυσμού θα πρέπει να χρησιμοποιήσουμε ένα μηχανισμό που θα μεταβιβάζει τις γενετικές πληροφορίες της προηγούμενης γενιάς στη νέα. Ένας εύκολος τρόπος είναι να δημιουργήσουμε μία οντότητα παιδί (child) η οποία θα είναι ένα ακριβές αντίγραφο του γονέα. Όμως στους παραδοσιακούς γενετικούς αλγορίθμους συνηθίζεται να επιλέγονται δύο γονείς εκ των οποίων προκύπτει μια μίξη γενετικού υλικού. Η μεταβίβαση των γονιδίων γίνεται με τον διαχωρισμό του γενετικού κώδικα δύο γονέων και την συνένωση τους ως νέο γονιδίωμα μιας νέας οντότητας απογόνου.



Εικόνα 14: Μεταβίβαση 50/50 γενετικού υλικού (DNA).

Ο διαχωρισμός μπορεί να γίνει είτε με τη λογική της κατά το ήμισυ διχοτόμησης του γενετικού υλικού των δύο γονέων (Εικόνα 14) είτε με τυχαία επιλογή σημείου διχοτόμησης (Εικόνα 15). Η δεύτερη μέθοδος προτιμάται μιας και αυξάνονται οι πιθανότητες για την εμφάνιση ποικιλομορφίας στο περιβάλλον μας εισάγοντας έτσι νέες λύσεις στο πρόβλημα.

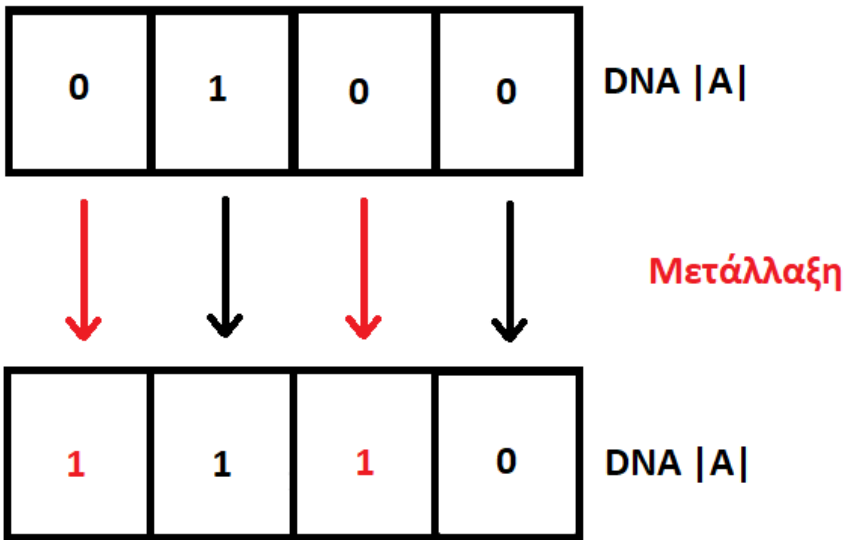


Εικόνα 15: Τυχαία επιλογή μεταβίβασης γενετικού υλικού (DNA).

4.2.3 Γενετική μετάλλαξη:

Στην περίπτωση που δεν υπάρχει αρκετή ποικιλομορφία στην αρχικό πληθυσμό είναι συχνά χρήσιμο να εισάγουμε μια επιπλέον παραλλαγή στο σύστημά μας, την μετάλλαξη.

Η μετάλλαξη αλλάζει ελαφρώς τα χαρακτηριστικά των οντοτήτων μας σε γονιδιακό επίπεδο. Η μετάλλαξη περιγράφεται ως ποσοστό, για παράδειγμα λέμε ότι ένας γενετικός αλγόριθμος έχει ποσοστό μετάλλαξης 1%. Πρακτικά σημαίνει ότι για κάθε οδηγία ενός γονιδίου υπάρχει 1% πιθανότητα μία συγκεκριμένη οδηγία να αντικατασταθεί πλήρως από μια τυχαία. Η μετάλλαξη μπορεί να μεταβάλλει μία ή περισσότερες γονιδιακές τιμές (Εικόνα 16).



Εικόνα 16: Γενετική μετάλλαξη γονιδίων.

Ο μηχανισμός της μετάλλαξης μπορεί να επηρεάσει σημαντικά το αποτέλεσμα της λύσης γι' αυτό το ποσοστό που καθορίζουμε δεν πρέπει να είναι πολύ υψηλό αλλιώς θα επικρατήσει η τυχαιότητα. Εάν επικρατεί η τυχαιότητα σε μεγάλο ποσοστό τότε τα γονίδια των προγόνων θα αντικαθίστανται πολύ συχνά από νέα, πράγμα που θα αναιρούσε την ίδια την εξελικτική διαδικασία αυτή καθαυτή.

Τέλος η μετάλλαξη εφαρμόζεται αφότου προηγηθεί η διαδικασία της διασταύρωσης στο γενετικό υλικό της οντότητας του απογόνου και επιδρά στα γονίδια αυτού. Το στάδιο της μετάλλαξης είναι το τελευταίο πριν την εκκίνηση της νέας γενιάς.

ΚΕΦΑΛΑΙΟ 5: ΕΠΕΞΗΓΗΣΗ ΤΟΥ ΚΩΔΙΚΑ

Genetic_Algorithm | Processing 3.5.4

File Edit Sketch Debug Tools Help

Java ▾

Genetic_Algorithm | Fish ▾

```
class Fish {

  PVector location; //Διάνυσμα θέσης.
  PVector velocity; //Διάνυσμα ταχύτητας.
  PVector acceleration; //Διάνυσμα επιτάχυνσης.

  int shape = 2; //Μεταβλητή που καθορίζει το μέγεθος της οντότητας.
  int maxSpeed = 5; //Μεταβλητή που καθορίζει τη μέγιστη ταχύτητα που μπορεί να αναπτύξει η ονότητα.

  //Δημιουργία οντότητας Fish.
  Fish() {
    //Αρχικοποίηση των διανυσμάτων.
    location = new PVector(width/3-100, height-120);
    velocity = new PVector();
    acceleration = new PVector();
  }

  //Μέθοδος που εμφανίζει την οντότητα στην οθόνη.
  void show() {
    //Η μέθοδος heading() υπολογίζει τη γωνία περιστροφής ενός διάνυσματος.
    float angle = velocity.heading() + PI/2;
    //Χρωματισμός του σχήματος.
    fill(255, 200);
    //Χρωματισμός του περιγράμματος.
    stroke(220);
    //Η Μεταφορά της αρχής των αξόνων στο κομμάτι του κώδικα που εμπεριέχεται ανάμεσα στις εντολές pushMatrix() και popMatrix().
    pushMatrix();
    //Μεταφορά της αρχής των αξόνων στο σημείο που θέλουμε να εμφανίζεται το σχήμα.
    translate(location.x, location.y);
    //Περιστρέφει το σχήμα της οντότητας ανάλογα με τη φορά που κινείται.
    rotate(angle);
    //Δημιουργία του σχήματος (ΤΡΙΓΩΝΟ) της οντότητας, ο κώδικας που περικλείεται στις εντολές beginShape() και endShape().
    beginShape(TRIANGLES);
    //Ένωση των παρακάτω κορυφών (vertex), για τον σχηματισμό του τριγώνου.
    vertex(0, -shape*2);
    vertex(-shape, shape*2);
    vertex(shape, shape*2);
    endShape();
    popMatrix();
  }

  //Μέθοδος που κινεί την οντότητα στην οθόνη.
  void move() {
    velocity.add(acceleration);//Εφαρμογή επιτάχυνσης στην ταχύτητα της οντότητας.
    velocity.limit(maxSpeed);//Περιορισμός του διανύσματος της ταχύτητας σε ένα όριο.
    location.add(velocity);//Εφαρμογή ταχύτητας στο διάνυσμα της θέσης.
  }
}
```

Εικόνα 17: Κλάση οντότητας Fish.

Ξεκινάμε με την δημιουργία της κλάσης της οντότητας που ονομάζουμε **Fish** (Εικόνα 17).

Η κάθε οντότητα έχει αρχικά ως χαρακτηριστικά:

- Διάνυσμα θέσης.
- Διάνυσμα ταχύτητας.
- Διάνυσμα επιτάχυνσης.
- Μεταβλητή που καθορίζει το μέγεθος της οντότητας.
- Μεταβλητή που καθορίζει τη μέγιστη ταχύτητα που μπορεί να αναπτύξει η οντότητα.

Στον κατασκευαστή την κλάσης γίνεται η αρχικοποίηση των διανυσμάτων, επιπλέον υλοποιούμε τις μεθόδους **show()** και **move()** οι οποίες αναλαμβάνουν την σχεδίαση της οντότητας και την εφαρμογή κίνησης σε αυτή αντίστοιχα.

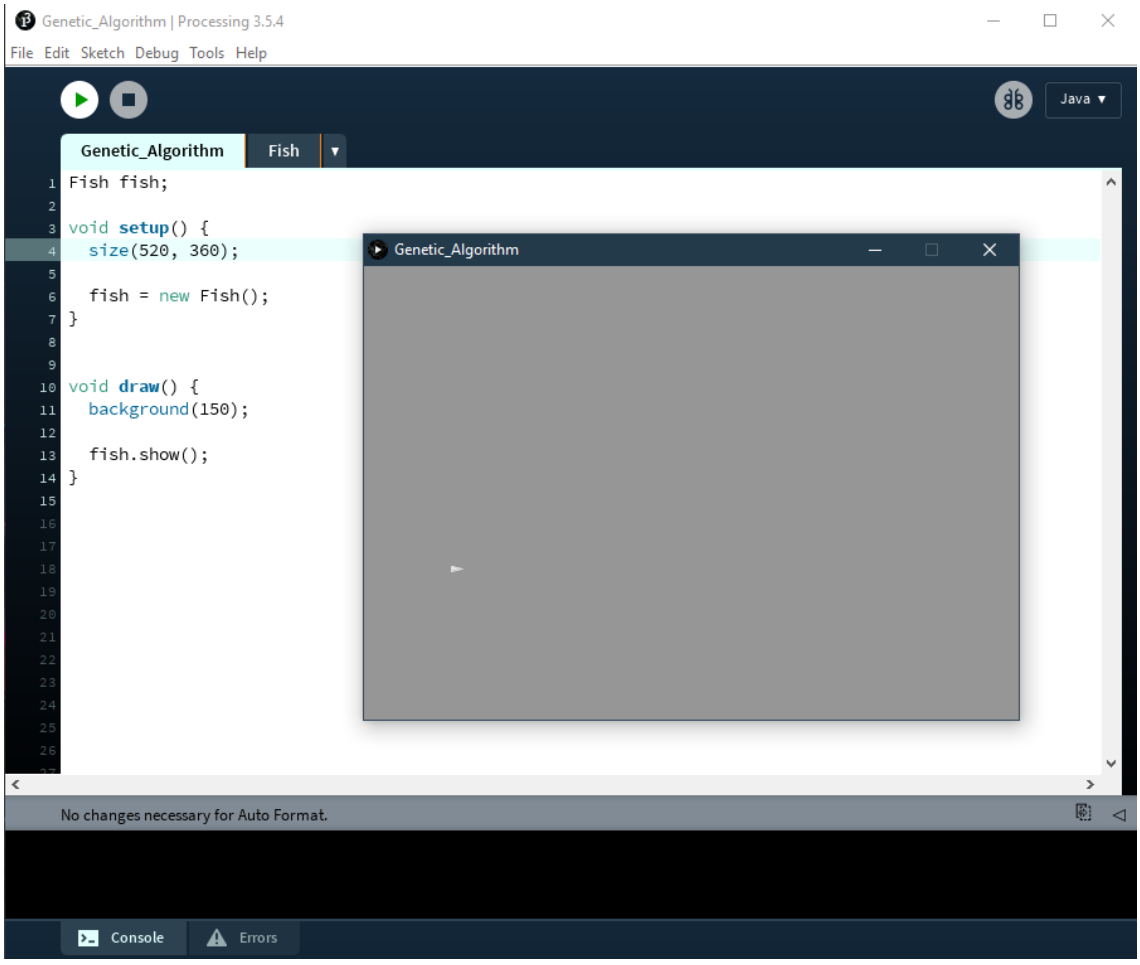
Στη γραμμή **29** του κώδικα παρατηρούμε την εντολή **translate** η οποία μεταφέρει την αρχή των αξόνων σε συγκεκριμένο σημείο του καμβά. Η αλλαγή της αρχής των αξόνων επηρεάζουν μόνο το κομμάτι του κώδικα το οποίο εμπεριέχεται ανάμεσα στις εντολές **pushMatrix()** και **popMatrix()**. Κάνουμε αυτή τη παρέμβαση διότι το σχήμα που έχουμε δημιουργήσει δεν έχει παραμέτρους συντεταγμένων για να καθορίσουμε που θα σχεδιαστεί οπότε μεταφέρουμε την αρχή των αξόνων εκεί που θέλουμε να σχεδιάσουμε την οντότητα. Αυτή η μεταφορά γίνεται τοπικά ώστε να μην επηρεαστεί άλλο κομμάτι του κώδικα.

Το σχήμα μας δημιουργείται εννοώντας τρία σημεία κορυφών (vertex) και καθορίζουμε το σχήμα τους σε τρίγωνο με την εντολή «**beginShape(TRIANGLES);»** που χρησιμεύει στη δημιουργία πιο περίπλοκων σχημάτων. Οι εντολές vertex περικλείονται ανάμεσα στις εντολές **beginShape()** και **endShape()** ώστε να καθοριστεί η αρχή και το τέλος των εντολών σχεδίασης του σχήματος.

Τέλος έχουμε συμπεριλάβει την εντολή **rotate** ώστε να περιστρέφουμε την οντότητα ανάλογα με το διάνυσμα της ταχύτητας. Η εντολή **velocity.heading()** καθορίζει την γωνία περιστροφής του διανύσματος της ταχύτητας, εφαρμόζουμε το αποτέλεσμα αυτής της εντολής στην rotate.

Για τη μέθοδο **move** χρησιμοποιούμε τις βασικές αρχές που έχουμε αναπτύξει σε προηγούμενο κεφάλαιο. Στη παρούσα φάση δεν υπάρχει κάποιο ερέθισμα για την κίνηση της οντότητας.

Η εκτέλεση του κώδικα μας δίνει τη παρακάτω απεικόνιση στον καμβά (Εικόνα 18).

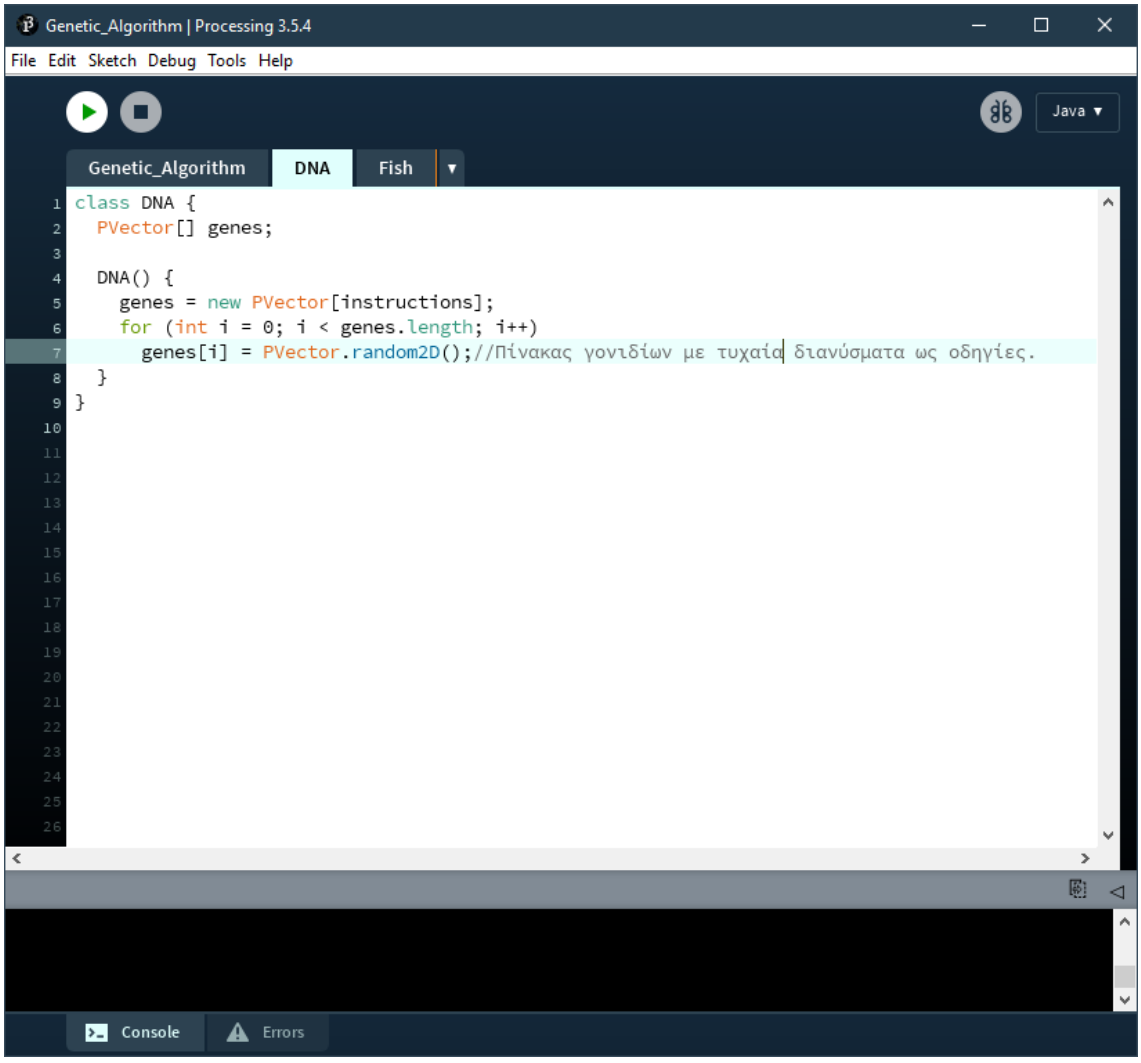


Εικόνα 18: Δοκιμή απεικόνισης της οντότητας.

Πλέον χρειαζόμαστε κάποιο ερέθισμα για να διαπιστώσουμε ότι η μέθοδος move δουλεύει σωστά. Όπως εξηγήσαμε στο προηγούμενο κεφάλαιο χρειαζόμαστε κάποιου είδους γονιδιακή πληροφορία. Στη περίπτωσή μας είναι οδηγίες πλοήγησης στο περιβάλλον του καμβά.

Δεν θα μπορούσαμε να εξετάσουμε όλες τις πτυχές της κίνησης των οντοτήτων σε όλο το φάσμα του περιβάλλοντος διότι αυτό θα απαιτούσε πάρα πολύ χρόνο μέχρι να πετύχουμε το βέλτιστο αποτέλεσμα. Αντίθετα θα μπορούσαμε να κάνουμε τυχαίες εικασίες για την κίνηση ενός αντικειμένου. Για αυτούς τους σκοπούς δημιουργούμε μια κλάση **DNA** που θα περιέχει το γενετικό υλικό της κάθε οντότητας (Εικόνα 19).

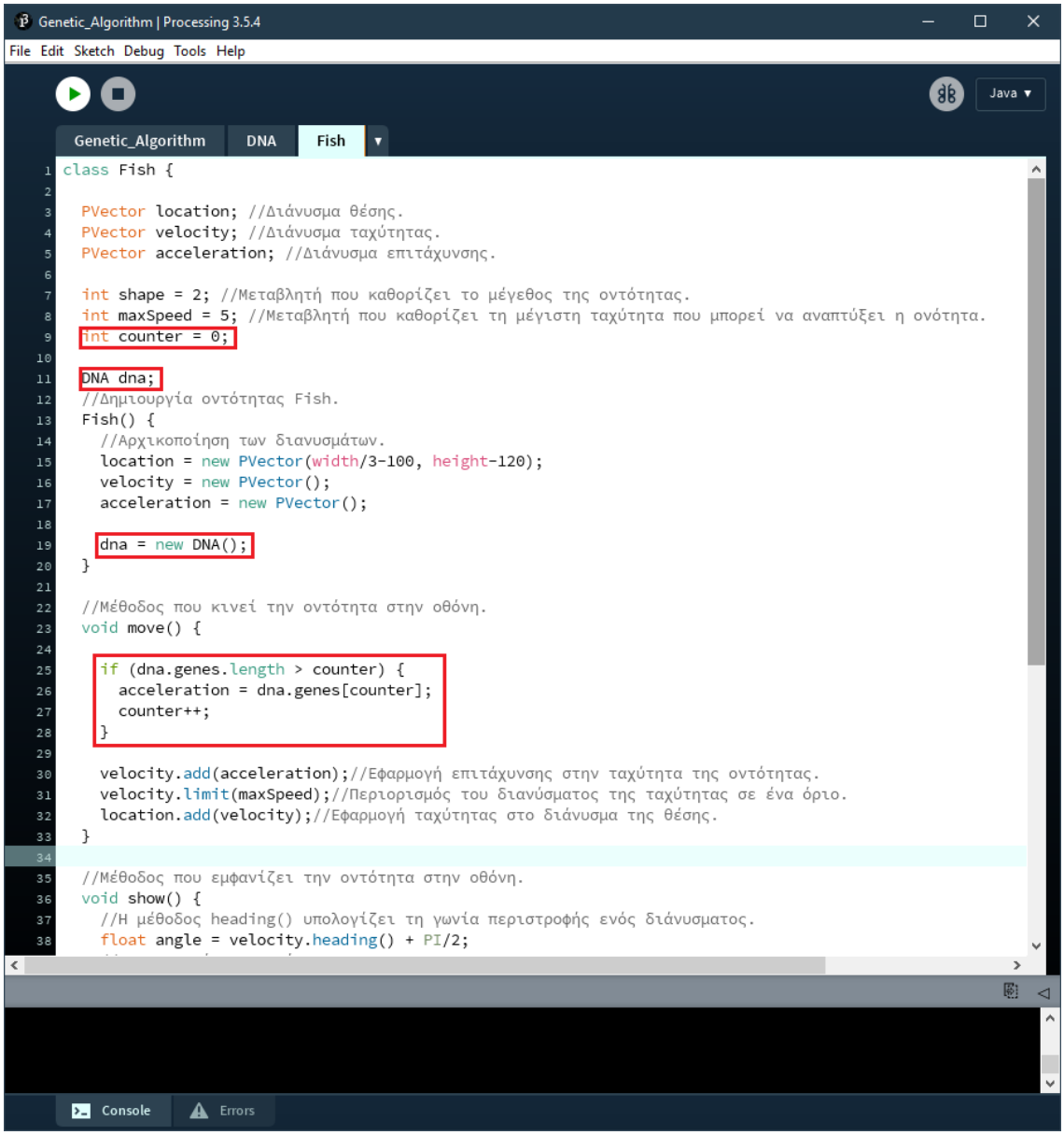
Η κάθε οντότητα έχει ένα εικονικό «DNA», ένα σύνολο ιδιοτήτων (γονίδια) που περιγράφουν συγκεκριμένες συμπεριφορές . Τα γονίδια θα είναι τυχαία διανύσματα που θα εφαρμόζονται ως οδηγίες στις οντότητές μας και είναι αποθηκευμένες στον πίνακα **genes** που έχει μέγεθος μία ποσότητα οδηγιών που καθορίζουμε εμείς από την κεντρική καρτέλα του προγράμματος (**Genetic_Algorithm**).



Εικόνα 19: Δημιουργία κλάσης DNA.

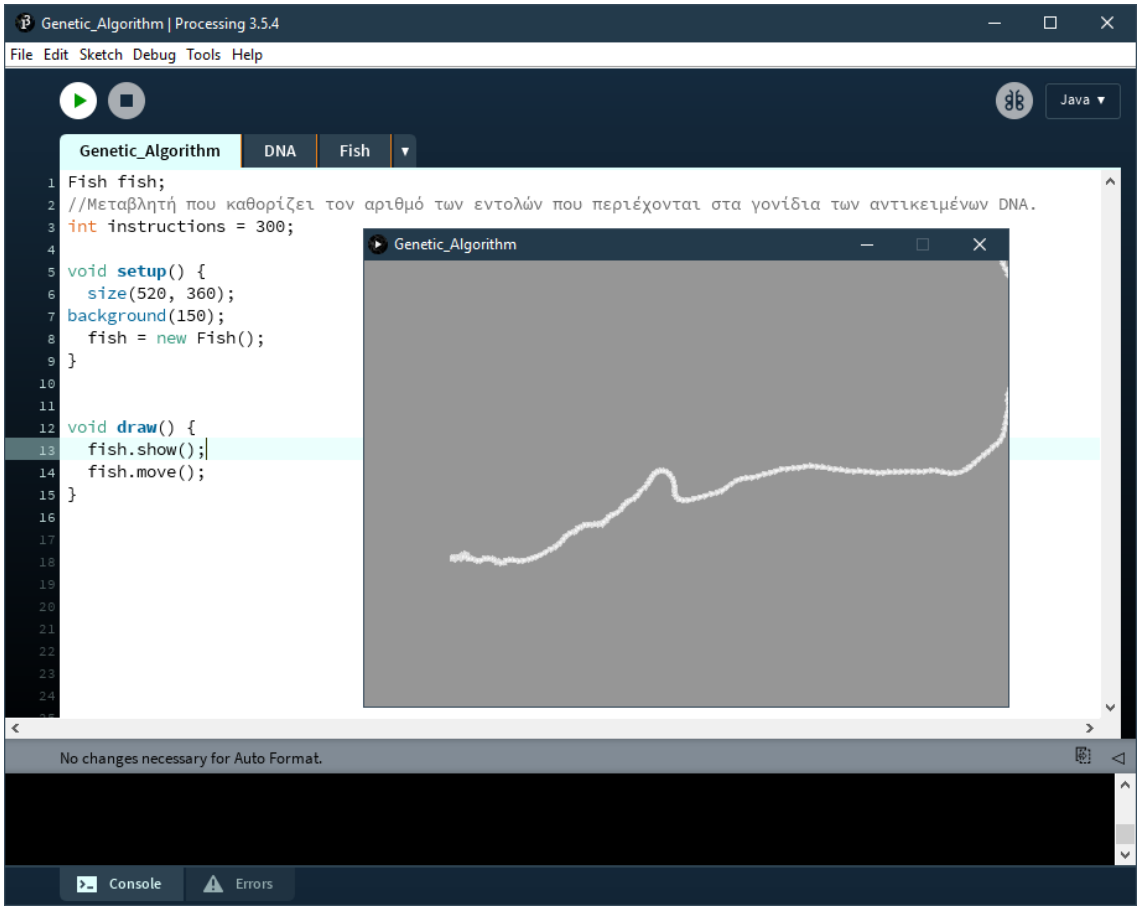
Οι οντότητές μας πρέπει να δράσουν σύμφωνα με το γενετικό υλικό που τους δίνεται, άρα στη κλάση Fish προσθέτουμε ένα αντικείμενο DNA.

Η μέθοδος move μετατρέπεται έτσι ώστε η ροή των οδηγιών από τα γονίδια να επηρεάζει τις τιμές της επιτάχυνσης όπου με τη σειρά της μεταβάλει την ταχύτητα και την θέση της οντότητας δίνοντάς της κίνηση (Εικόνα 20).Έχει επιστρατευτεί ένας μετρητής **counter** για την ελεγχόμενη ροή των δεδομένων (οδηγίες από τα γονίδια).



Εικόνα 20: Εφαρμογή οδηγιών DNA στην οντότητα.

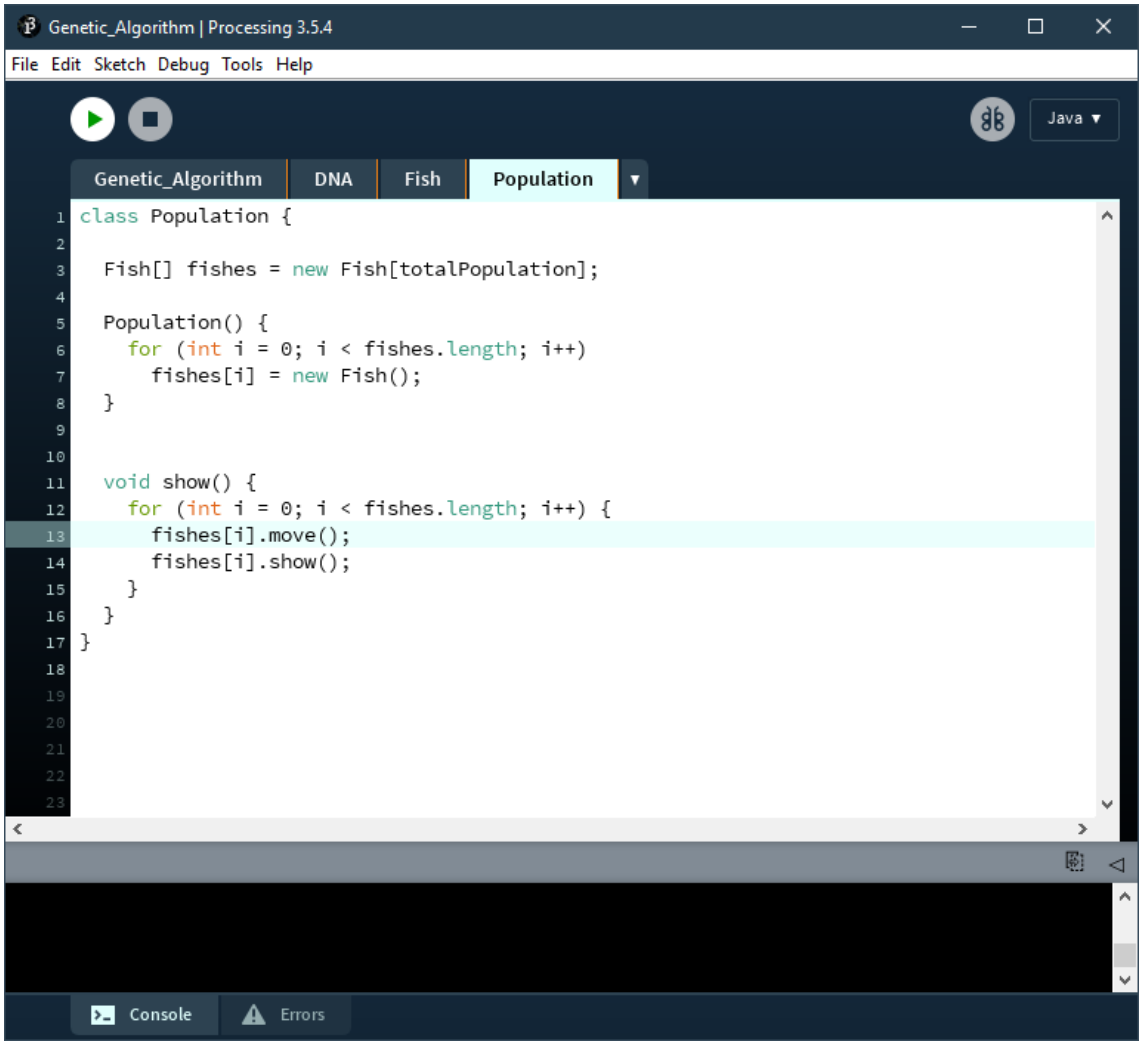
Παρακάτω στην κεντρική καρτέλα παρατηρούμε ότι έχουμε ορίσει τριακόσιες οδηγίες (**instructions**) ανά γενετικό υλικό (DNA). Με την εκτέλεση φαίνεται ότι η οντότητα κινείται στον χώρο και εν τέλει βγαίνει εκτός ορίων.



Εικόνα 21: Εκτέλεση του προγράμματος.

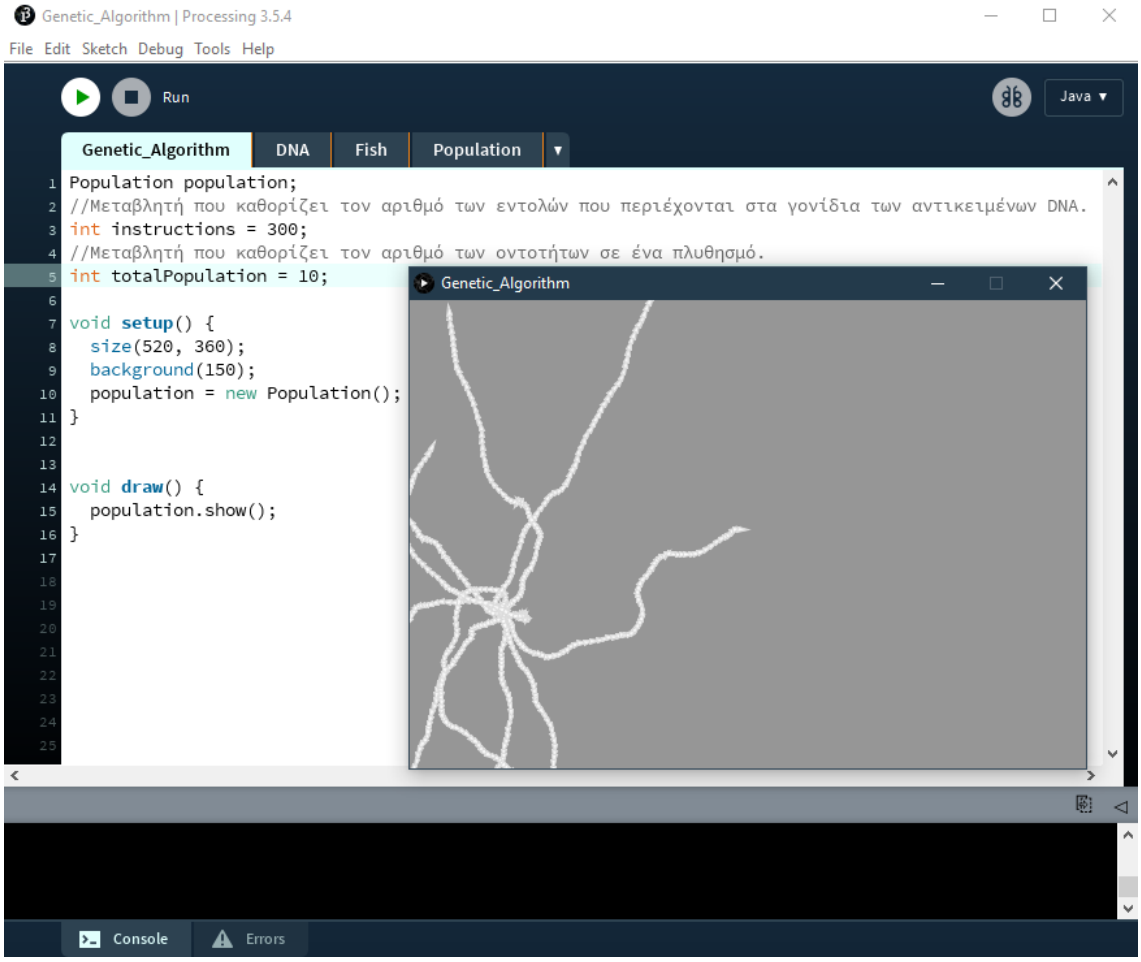
Εφόσον λειτουργεί η προσομοίωσή μας για μία οντότητα μπορούμε πλέον να μοντελοποιήσουμε έναν πληθυσμό από αυτές.

Δημιουργούμε μία κλάση πληθυσμού **Population** στην οποία δημιουργούμε έναν πίνακα που θα φιλοξενεί μια γενιά οντοτήτων και πλέον καλούμε τις μεθόδους move και show των οντοτήτων Fish μέσω της μεθόδου show της Population (Εικόνα 22).



Εικόνα 22: Δημιουργία κλάσης Population.

Εκτελούμε δοκιμαστικά το πρόγραμμα αφού πρώτα έχουμε καθορίσει τον αριθμό του πλήθους των οντοτήτων ορίζοντας μια μεταβλητή **totalPopulation** και καλώντας τη μέθοδο show της Population για την απεικόνιση αυτών (Εικόνα 23).



Εικόνα 23: Εκτέλεση προγράμματος.

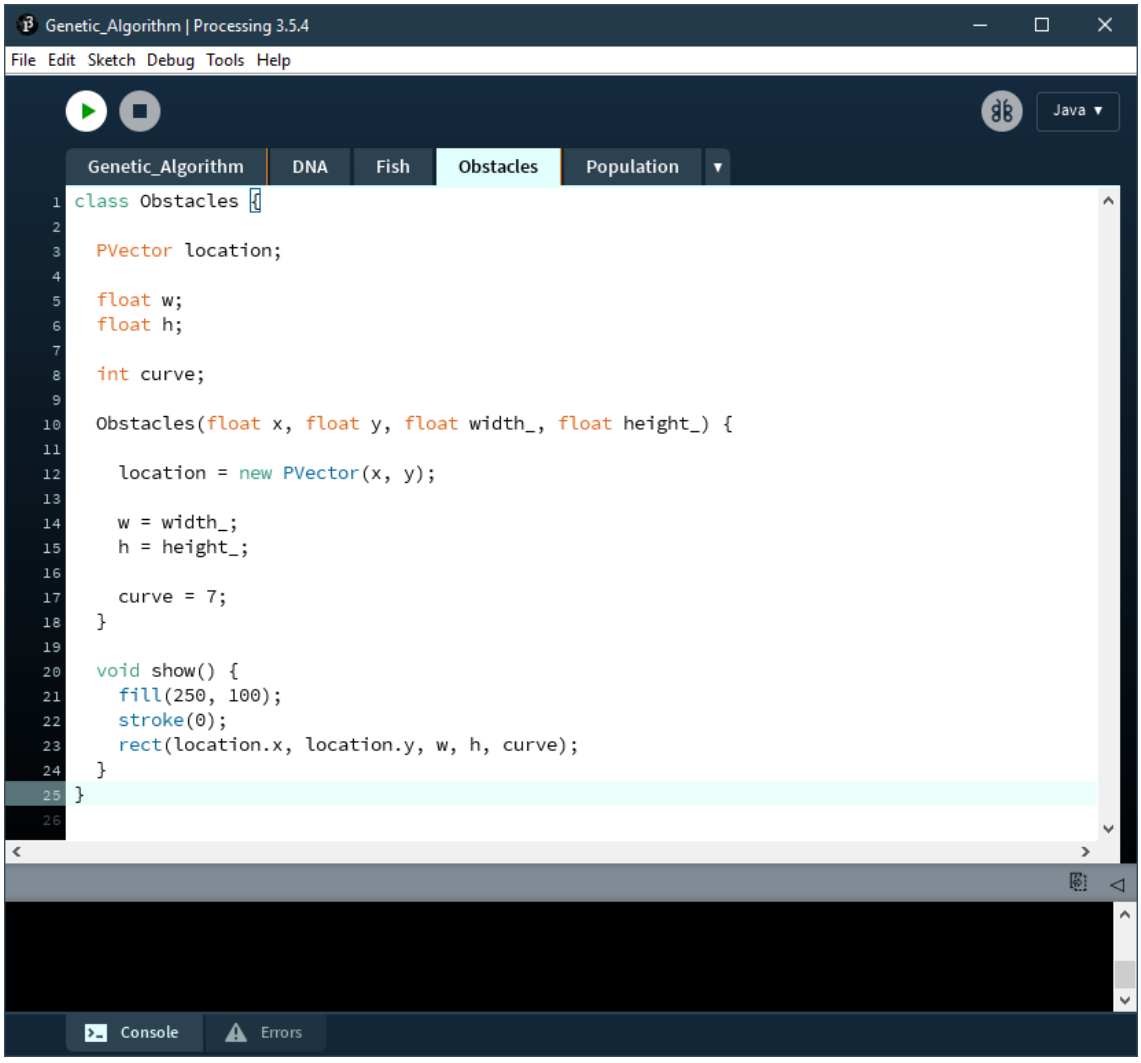
Οι οντότητες κινούνται τυχαία και ελεύθερα στον χώρο, στη προσομοίωση μας θέλουμε ιδανικά η κάθε οντότητα να σταματά πριν βγει εκτός ορίων, όταν έρθει σε επαφή με κάποιο εμπόδιο ή σε περίπτωση που δεν έχει άλλες οδηγίες να ακολουθήσει.

Για την κάλυψη της περίπτωσης των εμποδίων θα δημιουργήσουμε μια κλάση που θα περιγράφει το σχήμα του εμποδίου και θα το εμφανίζει στην οθόνη. Αυτή η κλάση θα ονομαστεί **Obstacles** και θα αφορά σχήματα ορθογώνιων παραλληλογράμμων (Εικόνα 24).

Ένα αντικείμενο τύπου Obstacles αποτελείται από:

- Το διάνυσμα θέσης του.
- Το πλάτος του και,
- Το ύψος του.

Σαν επιπλέον χαρακτηριστικό έχει προστεθεί και η μεταβλητή **curve** που καθορίζει την ομαλότητα των αιχμών των γωνιών του σχήματος με σκοπό μια ομορφότερη απεικόνιση.

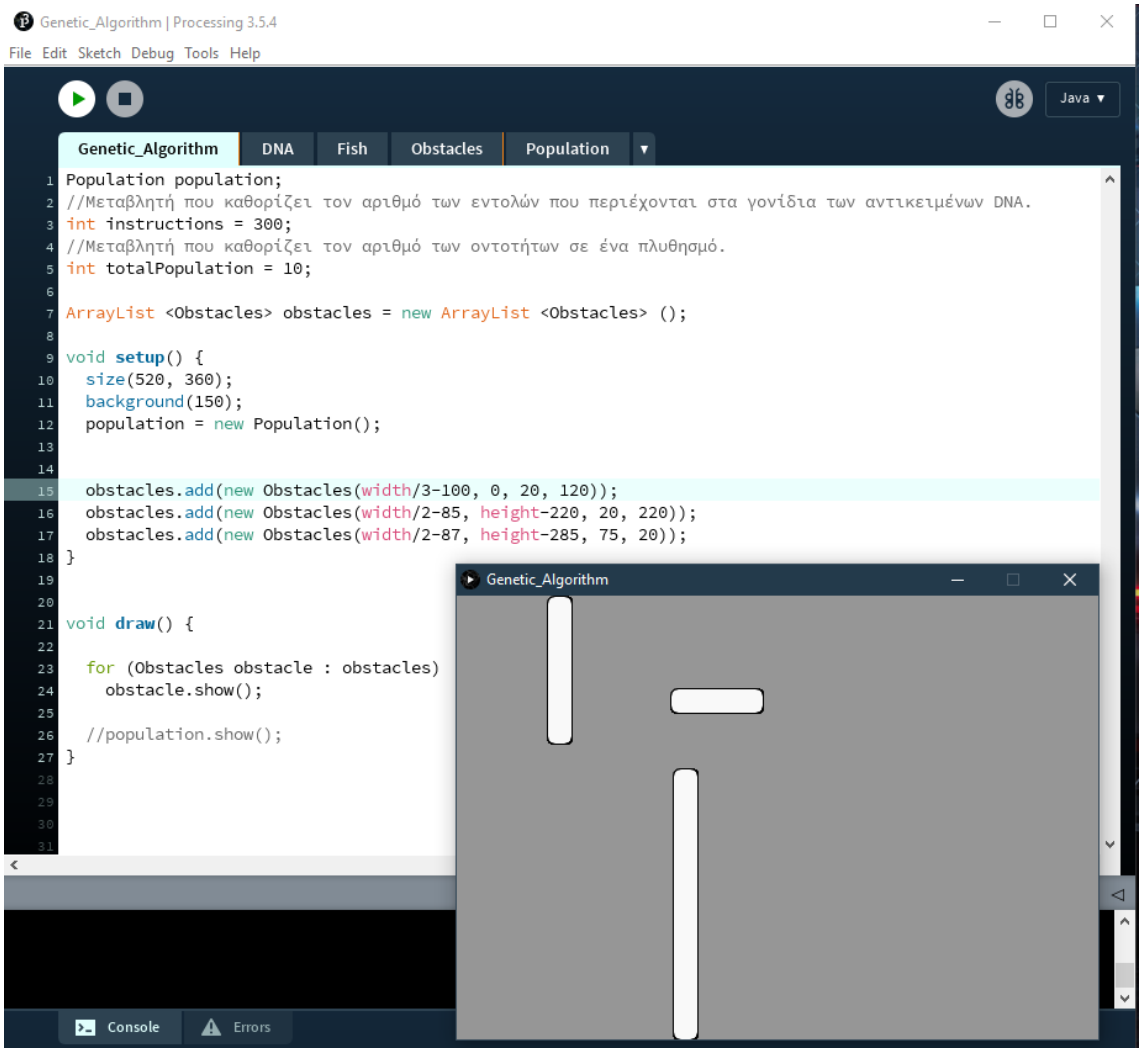


Εικόνα 24: Δημιουργία κλάσης Obstacles.

Η μέθοδος show αναλαμβάνει να εμφανίσει το αντικείμενο στον καμβά χρησιμοποιώντας την μέθοδο **rect()** για τη δημιουργία του σχήματος εφαρμόζοντας τις μεταβλητές που έχουμε ορίσει από το κατασκευαστή της κλάσης.

Για την αναπαράσταση πολλών εμποδίων υλοποιήσαμε μια δυναμική λίστα (**ArrayList**) που θα φιλοξενεί τα αντικείμενα των εμποδίων.

Τέλος δημιουργούμε τρία αντικείμενα εμποδίων και τα εμφανίζουμε στην οθόνη μέσω μιας ενισχυμένης δομής επανάληψης **for** (**enhanced for loop**) (Εικόνα 25).



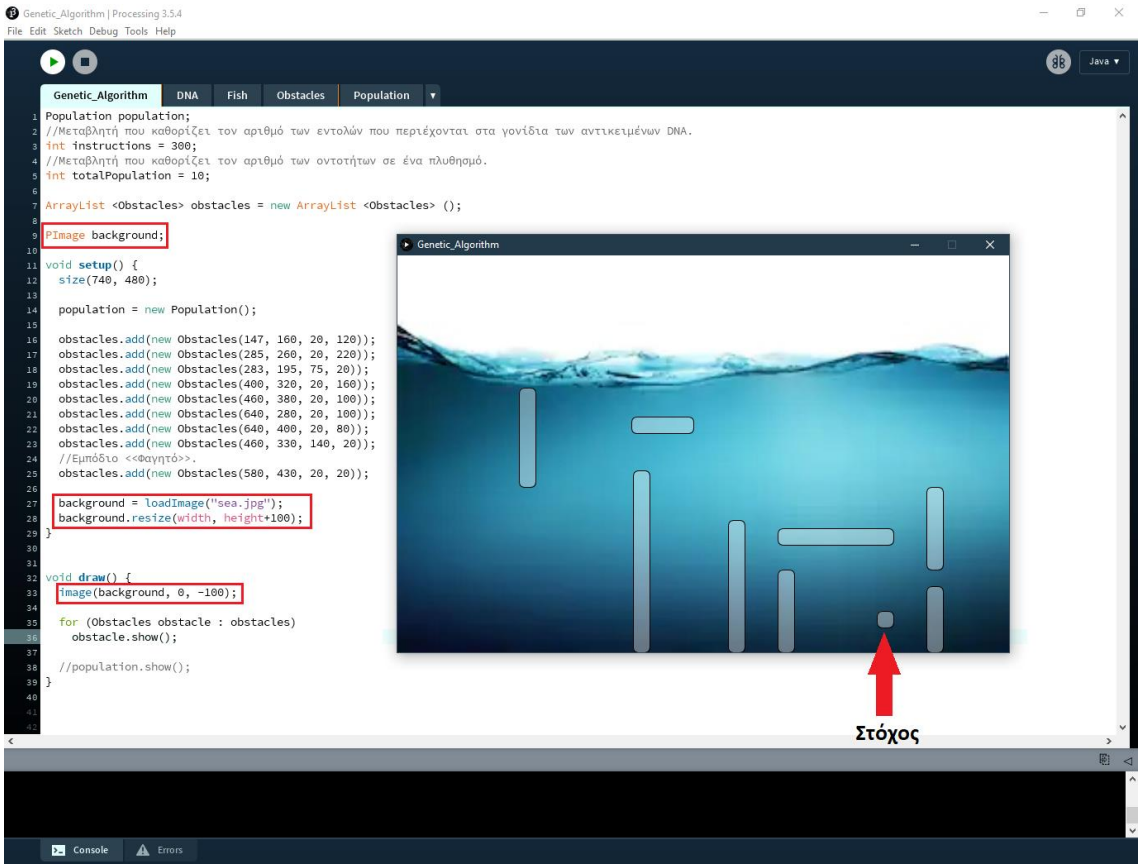
Εικόνα 25: Εμφάνιση των εμποδίων.

Επιπλέον αντικαθιστούμε το γκρι background με μία εικόνα που αρμόζει καλύτερα στη φύση του παραδείγματος (Εικόνα 26). Δημιουργούμε λοιπόν ένα αντικείμενο **PImage** που μας επιτρέπει τον χειρισμό εικόνων, έτσι φορτώνουμε την εικόνα μας μέσω της εντολής **loadImage()** και αλλάζουμε το μέγεθός της ώστε να εφαρμόζει σωστά στον καμβά μας.

Τέλος με την εντολή **image()** εμφανίζουμε την εικόνα μας ως νέο background σε κάθε καρέ (frame) εφόσον βρίσκεται στο κομμάτι του κώδικα της μεθόδου draw.

Ακόμη προσθέτουμε ένα νέο εμπόδιο που θα αποτελέσει τον στόχο για τις οντότητές μας.

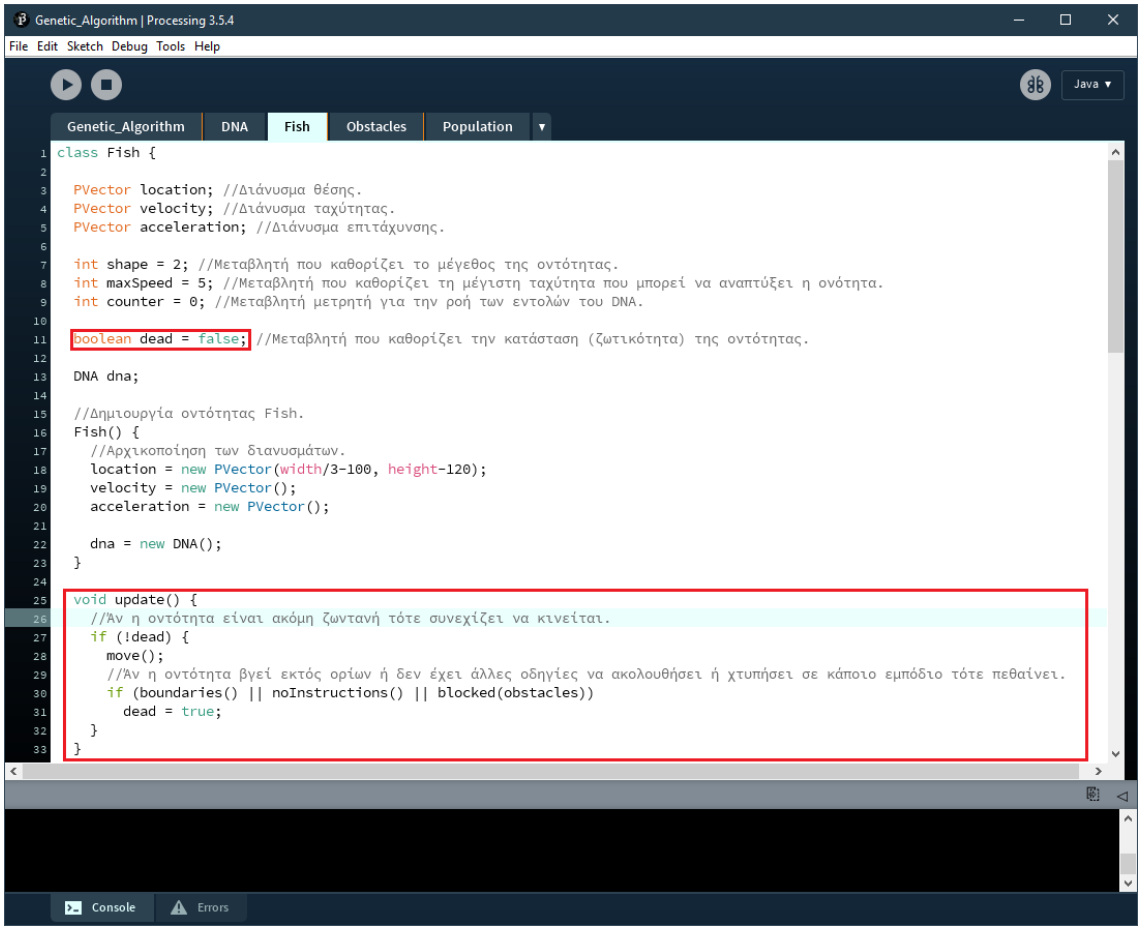
Με την εκ νέου εκτέλεση παρατηρούμε ότι οι αλλαγές μας έχουν τεθεί σε εφαρμογή.



Εικόνα 26: Τελικός σχεδιασμός.

Όπως προείπαμε ιδανικά θέλουμε να θέσουμε συγκεκριμένα όρια στις οντότητές μας. Κάθε φορά που η οντότητα πρόκειται να βγεί εκτός ορίων ή δεν έχει άλλες οδηγίες να ακολουθήσει ή χτυπήσει σε κάποιο εμπόδιο τότε θα λέμε ότι πεθαίνει (σταματάει να κινείται).

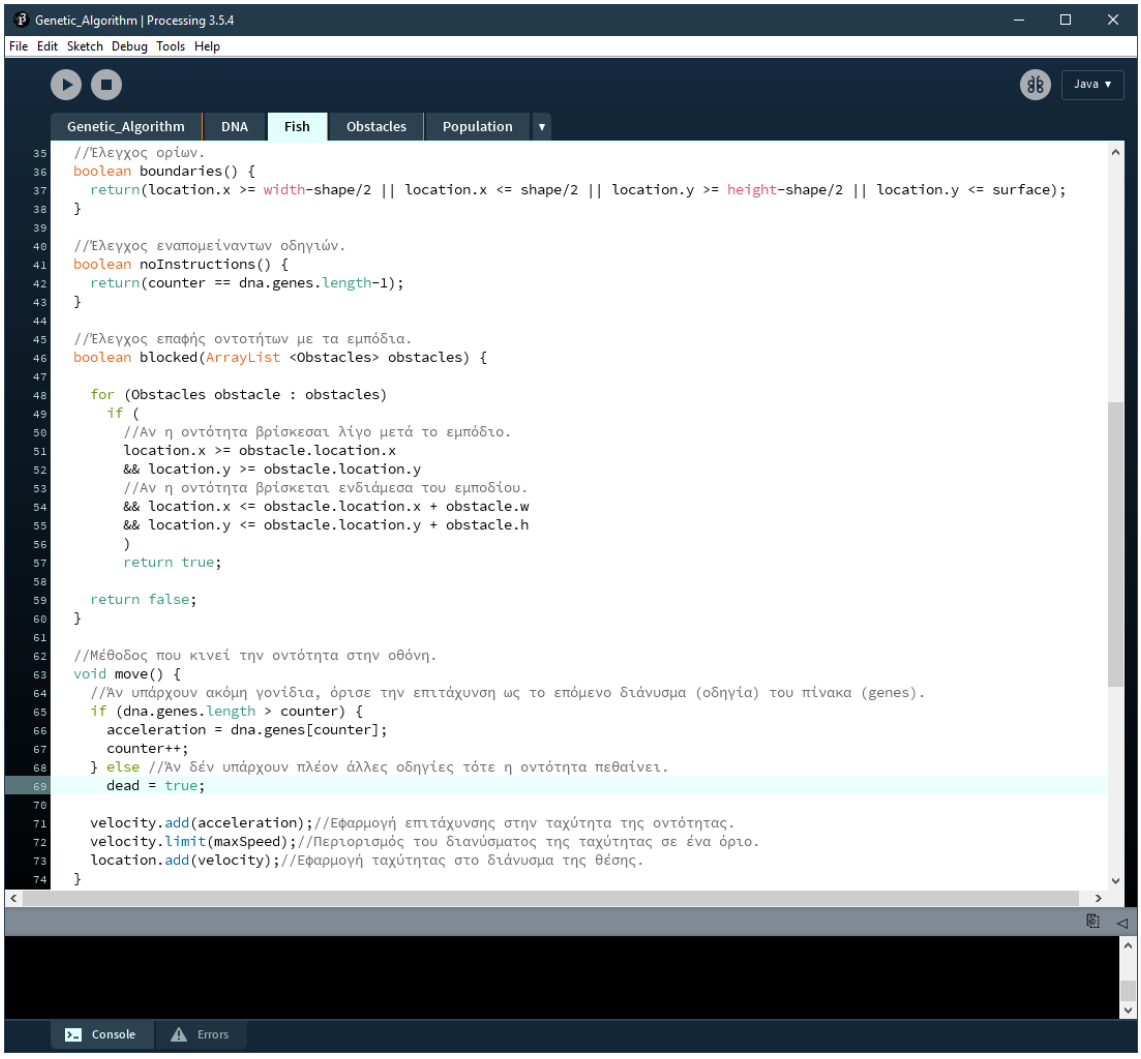
Οπότε εισάγουμε στη κλάση Fish μία μεταβλητή **dead** που καθορίζει την κατάσταση (ζωτικότητα) της οντότητας. Επιπλέον δημιουργήσαμε μία νέα μέθοδο, την **update().** Η update κάθε στιγμή κινεί και ελέγχει τις ζωντανές οντότητες, αν κάποια οντότητα δεν συμμορφώνεται με τις παραπάνω παραδοχές τότε την θεωρεί νεκρή (Εικόνα 27).



Εικόνα 27: Μέθοδος update.

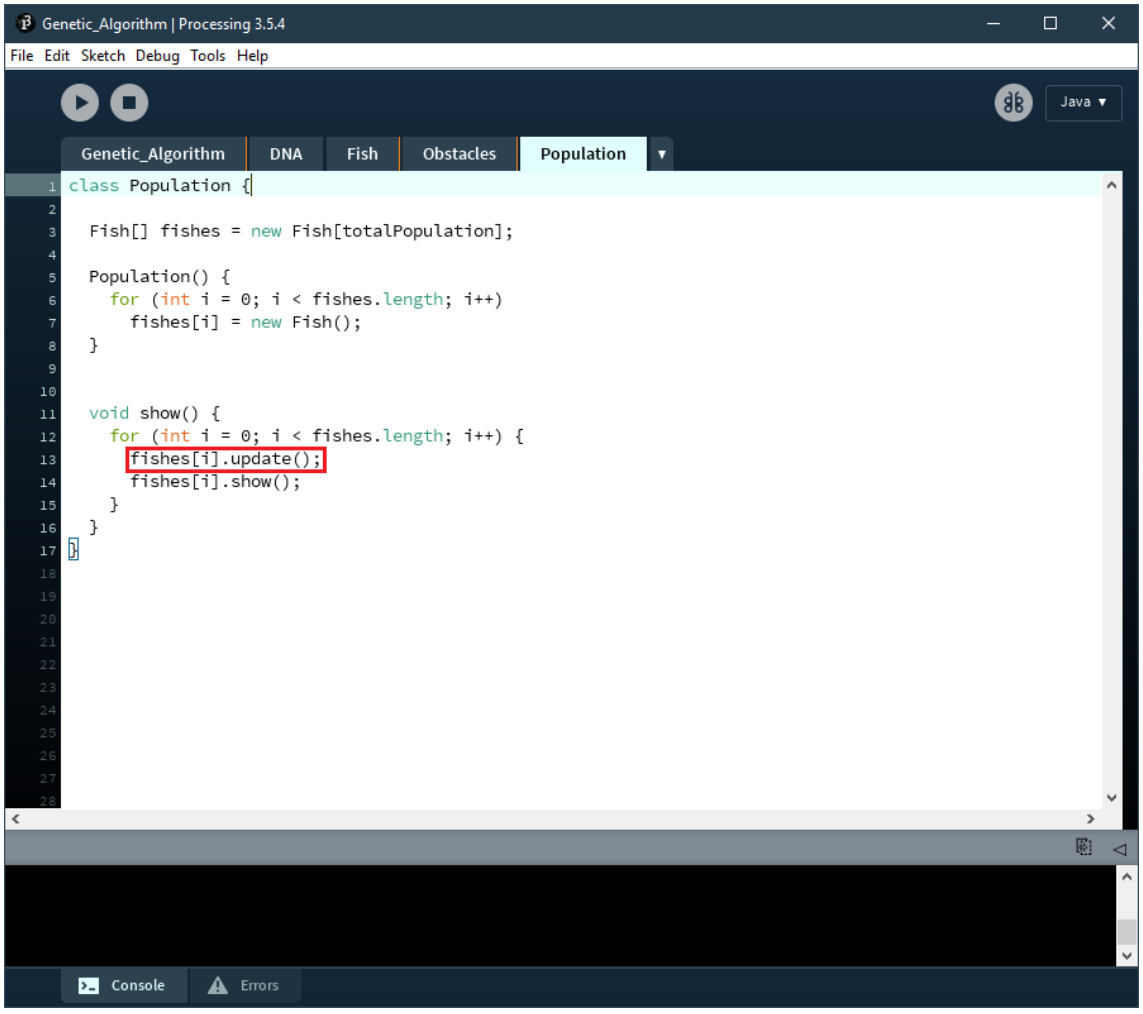
Η μέθοδος **boundaries()**, ελέγχει αν η οντότητα βρίσκεται εντός ορίων, η μεταβλητή **surface** υποδηλώνει την επιφάνεια της θάλασσας στην εικόνα του background. Η μεταβλητή surface έχει οριστεί στην κύρια καρτέλα ως το νέο πάνω όριο του παραθύρου με τιμή **120** pixel κάτω από το πρωτότυπο όριο (Εικόνα 30).

Η μέθοδος **noInstructions()**, ελέγχει αν υπάρχουν εναπομείνασες οδηγίες, ενώ η μέθοδος **blocked()** ελέγχει αν η οντότητα έχει έρθει σε επαφή με κάποιο εμπόδιο (εικόνα 28).



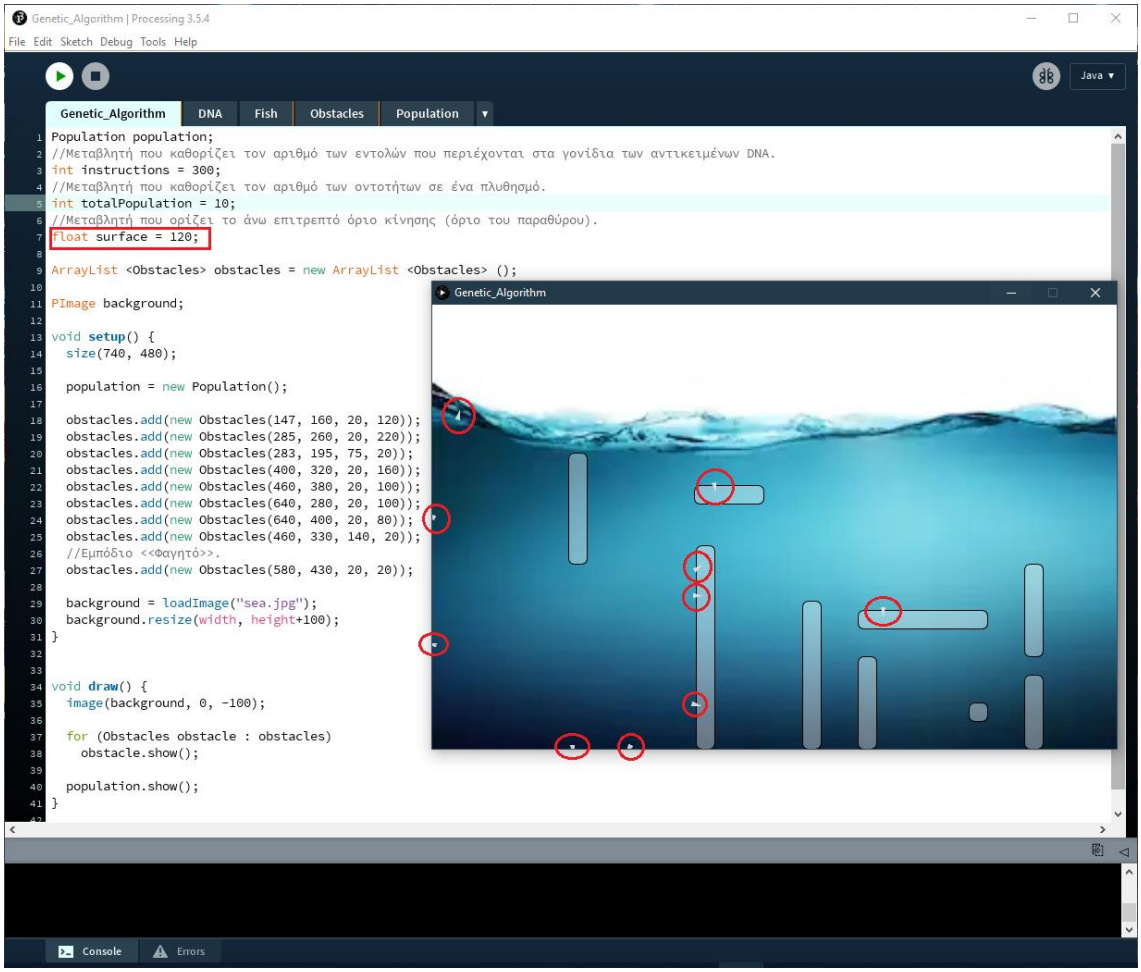
Εικόνα 28: Μέθοδοι boundaries, noInstructions και blocked.

Η μέθοδος move καλείται πλέον από την μέθοδο update (Εικόνα 27) αφού προηγηθεί κάποιος έλεγχος, οπότε στη κλάση Population θα χρειαστεί να καλέσουμε την **update** αντί της move (Εικόνα 29).



Εικόνα 29: Αλλαγή στην μέθοδο show της κλάσης Population.

Δοκιμάζουμε τη συμπεριφορά των οντοτήτων και παρατηρούμε ότι συμμορφώνονται με όλες τις παραδοχές που έχουμε ορίσει για την προσομοίωσή μας (Εικόνα 30).

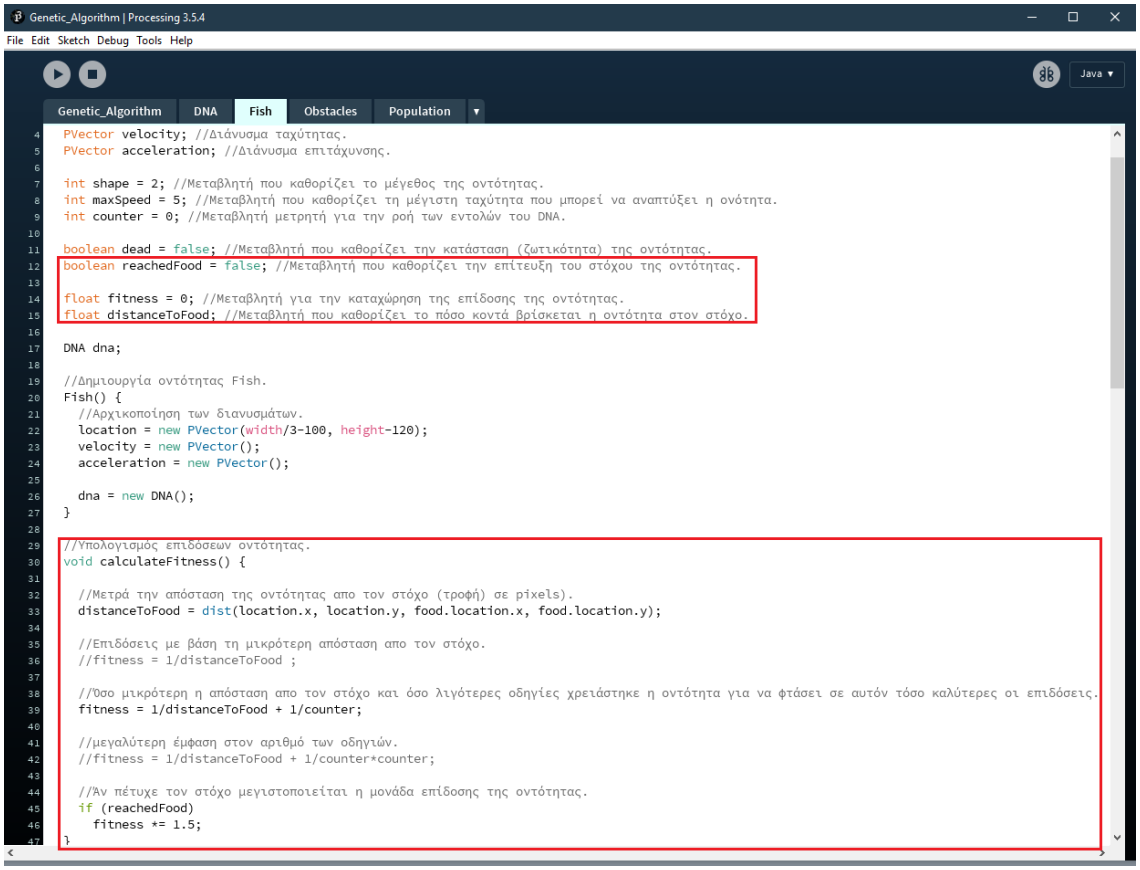


Εικόνα 30: Δοκιμή συμπεριφοράς των οντοτήτων.

Σε αυτή τη φάση ξεκινά η υλοποίηση του γενετικού αλγορίθμου, δεν μας ενδιαφέρει τόσο μια ακριβής προσομοίωση της εξέλιξης αλλά περισσότερο για τις μεθόδους που θα εφαρμόζουν εξελικτικές στρατηγικές.

Οι οντότητές μας κινούνται βάσει τυχαίων οδηγιών, αν υποθέσουμε λοιπόν ότι κάθε φορά που φτάνουμε κοντύτερα η μακρύτερα από το στόχο μας μια συνάρτηση θα αξιολογούσε τις επιδόσεις της οντότητας ανάλογα με το πόσο κοντά βρίσκεται στην επίτευξη του δοθέντος στόχου, έτσι θα φτάναμε πιο σύντομα στη λύση με αποτέλεσμα να εξελίξουμε τον πληθυσμό μας προς αυτή τη κατεύθυνση . Όπως γνωρίζουμε από το προηγούμενο κεφάλαιο την αξιολόγηση αυτή θα αναλάβει για εμάς μία συνάρτηση αξιολόγησης επιδόσεων (fitness function).

Άρα στη κλάση Fish θα υλοποιήσουμε μία μέθοδο υπολογισμού της επίδοσης της οντότητας, αυτή θα είναι η **calculateFitness()** (Εικόνα 31).



Εικόνα 31: Υπολογισμός επιδόσεων.

Στην κλάση μας εισάγουμε επίσης τις μεταβλητές **reachedFood,** που καθορίζει την επίτευξη του στόχου της οντότητας, **fitness** για την καταχώρηση της επίδοσης της οντότητας και **distanceToFood** που καθορίζει το πόσο κοντά βρίσκεται η οντότητα στον στόχο.

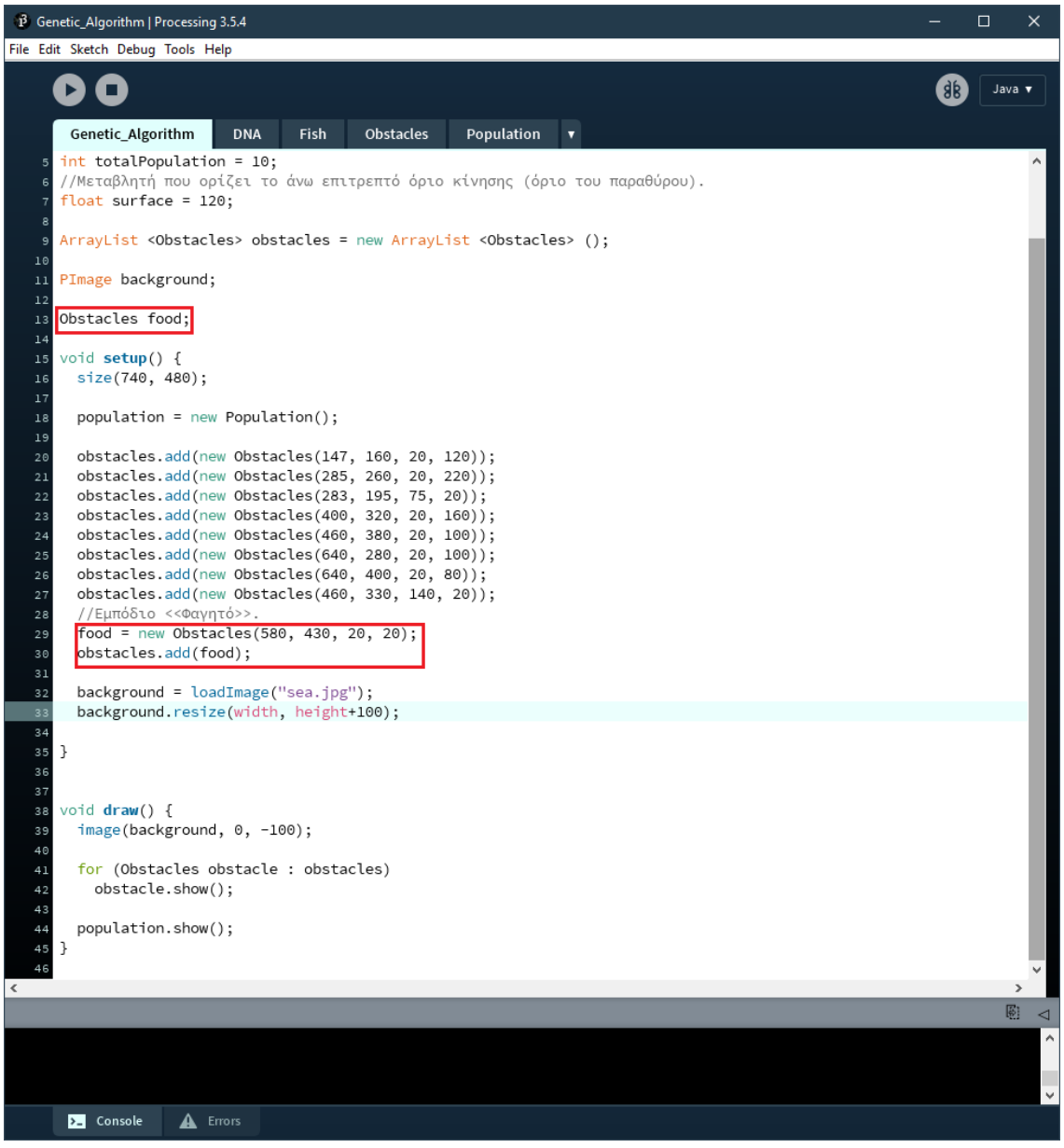
Η distanceToFood υπολογίζεται με τη βοήθεια της εντολής **dist()** που δίνει ως αποτέλεσμα την απόσταση σε pixel μεταξύ δύο διανυσμάτων στους άξονες X/Y. Στη συγκεκριμένη περίπτωση μεταξύ του διανύσματος της θέσης της οντότητας με αυτού του στόχου.

Ο στόχος δεν είναι πλέον ένα ανώνυμο αντικείμενο τύπου Obstacles έτσι ώστε να έχουμε πρόσβαση στο διάνυσμα της θέσης του δια της αναφοράς του, το όνομα αυτής θα είναι food (Εικόνα 32).

Η τιμή της επίδοσης καθορίζεται από την εντολή εξίσωσης «**fitness = 1/distanceToFood + 1/counter;**», δηλαδή όσο μικρότερη η απόσταση από τον στόχο και όσο λιγότερες οδηγίες χρειάστηκε η οντότητα για να φτάσει σε αυτόν τόσο καλύτερες οι επιδόσεις.

Θυμίζουμε ότι ο μετρητής counter είναι ο αριθμός των οδηγιών που έχει καταναλώσει μία οντότητα μέχρι το σημείο που έχει φτάσει ανεξάρτητα με την κατάσταση ζωτικότητάς της. Άρα όσο λιγότερα βήματα έκανε για να φτάσει στον στόχο τόσο μεγαλύτερη η επιβράβευση. Με αυτόν τον τρόπο οι οντότητες μετά από την εξέλιξη αρκετών γενεών βρίσκουν τη βέλτιστη διαδρομή εάν έχει επιλεγεί μια μη αποδοτική προγενέστερα. Μετά από κάθε γενιά, η μέση απόσταση που διανύθηκε μειώνεται γιατί τα καλά χαρακτηριστικά γίνονται όλο και πιο κοινά.

Τέλος αν κάποια οντότητα πέτυχε τον στόχο της δέχεται μία αύξηση στην τιμή της επίδοσής της.



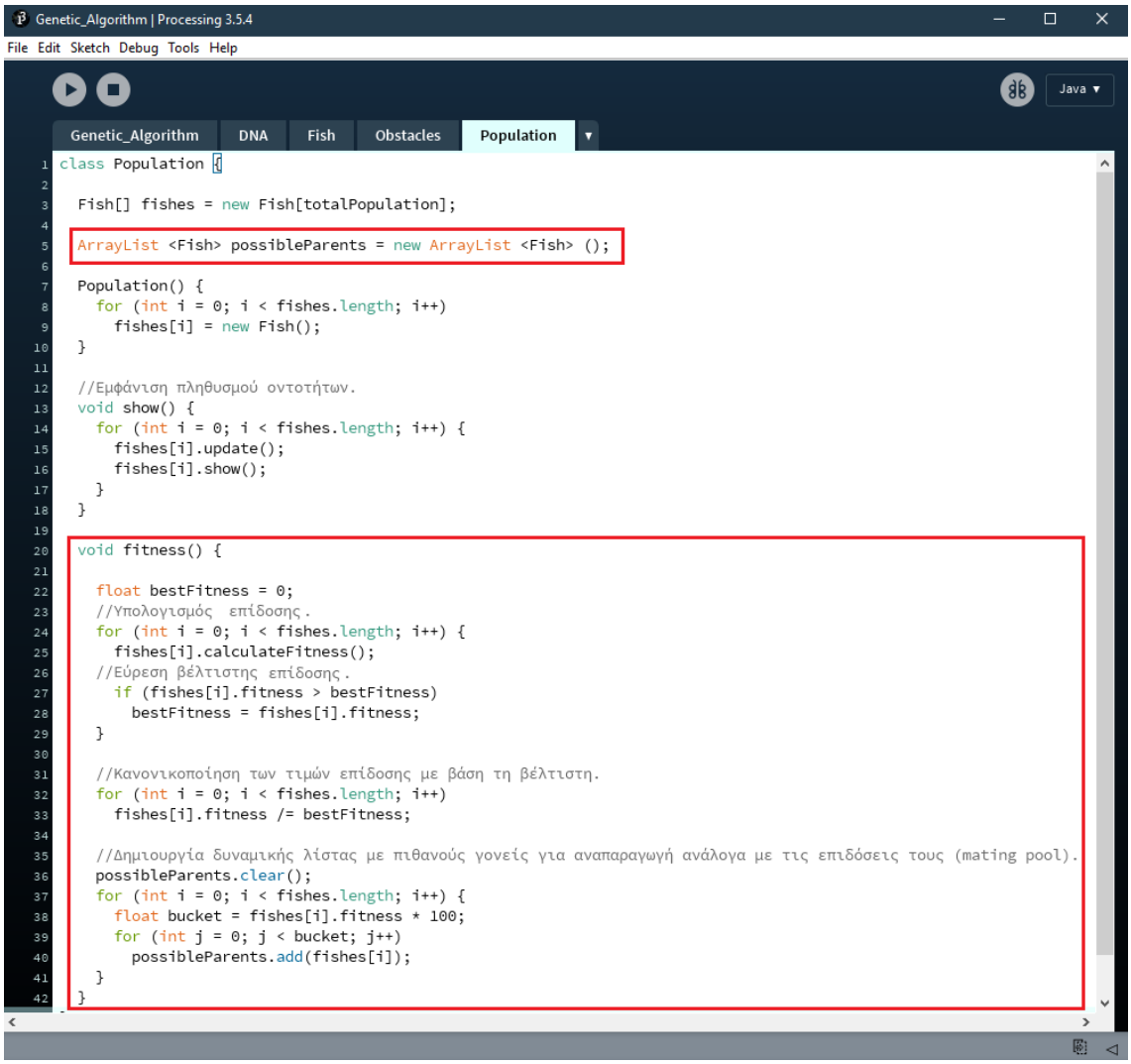
Εικόνα 32: Ορισμός αντικειμένου στόχου.

Μόλις υπολογιστεί η καταλληλόλητα για όλα τα μέλη του πληθυσμού, μπορούμε στη συνέχεια να επιλέξουμε ποια μέλη είναι κατάλληλα για να γίνουν γονείς και να τα τοποθετήσουμε σε μια ομάδα ζευγαρώματος (**mating pool**).

Η μέθοδος **fitness()** της κλάσης Population υλοποιεί την ομάδα ζευγαρώματος για κάθε γενιά. Πρώτα υπολογίζονται οι επιδόσεις όλων των οντοτήτων, έπειτα καταχωρείται η βέλτιστη επίδοση και κανονικοποιούμε τις επιδόσεις των οντοτήτων βάσει αυτής ώστε να χειριζόμαστε τιμές επιδόσεων από **0** έως **1** (Εικόνα 33).

Δημιουργούμε λοιπόν μία δυναμική λίστα που ονομάσαμε **possiblePartents** που θα φιλοξενεί οντότητες με πιθανότητες (ανάλογες με τις επιδόσεις τους) να επιλεχθούν για αναπαραγωγή.

Ο κώδικας από τη γραμμή **37** έως την **41** καταχωρεί στη λίστα μας την κάθε οντότητα τόσες φορές όσες η τιμή επίδοσής της πολλαπλασιασμένη με το εκατό. Αυτή η τακτική αυξάνει τις πιθανότητες να επιλεγεί για αναπαραγωγή μια οντότητα με πολύ καλές επιδόσεις. Για παράδειγμα, εφόσον χειριζόμαστε τιμές από **0** έως **1** τότε αν μία οντότητα έχει επίδοση **0,34** η πολλαπλασιασμένη τιμή της είναι **34,** άρα η συγκεκριμένη οντότητα θα τοποθετηθεί στη λίστα μας 34 φορές με 34% πιθανότητα να επιλεχθεί για αναπαραγωγή .



Εικόνα 33: Δημιουργία ομάδας ζευγαρώματος (mating pool).

Πλέον εφόσον έχουμε δημιουργήσει μια ομάδα ζευγαρώματος μένει οι οντότητες μας να μεταβιβάσουν το DNA τους σε μια νέα γενιά οντοτήτων (Εικόνα 34). Οπότε στη κλάση Population υλοποιούμε τη μέθοδο **birth()** η οποία αναλαμβάνει να δημιουργήσει οντότητες απογόνων.

Πρώτα δημιουργούμε έναν πίνακα οντοτήτων Fish με ονομασία **newFishes** και μέγεθος ίδιο με του αρχικού πληθυσμού που θα φιλοξενεί τις οντότητες των απογόνων (νέα γενιά). Έπειτα δημιουργούμε αντικείμενα γονέων τύπου DNA με ονομασίες **parentA** και **parentB** με σκοπό την αποκόμιση των γονιδίων δύο τυχαίων γονέων. Συνεχίζουμε δημιουργώντας έναν πίνακα διανυσμάτων **child_genes** με μέγεθος ίδιο με τις οδηγίες (instructions) της πρώτης γενιάς.

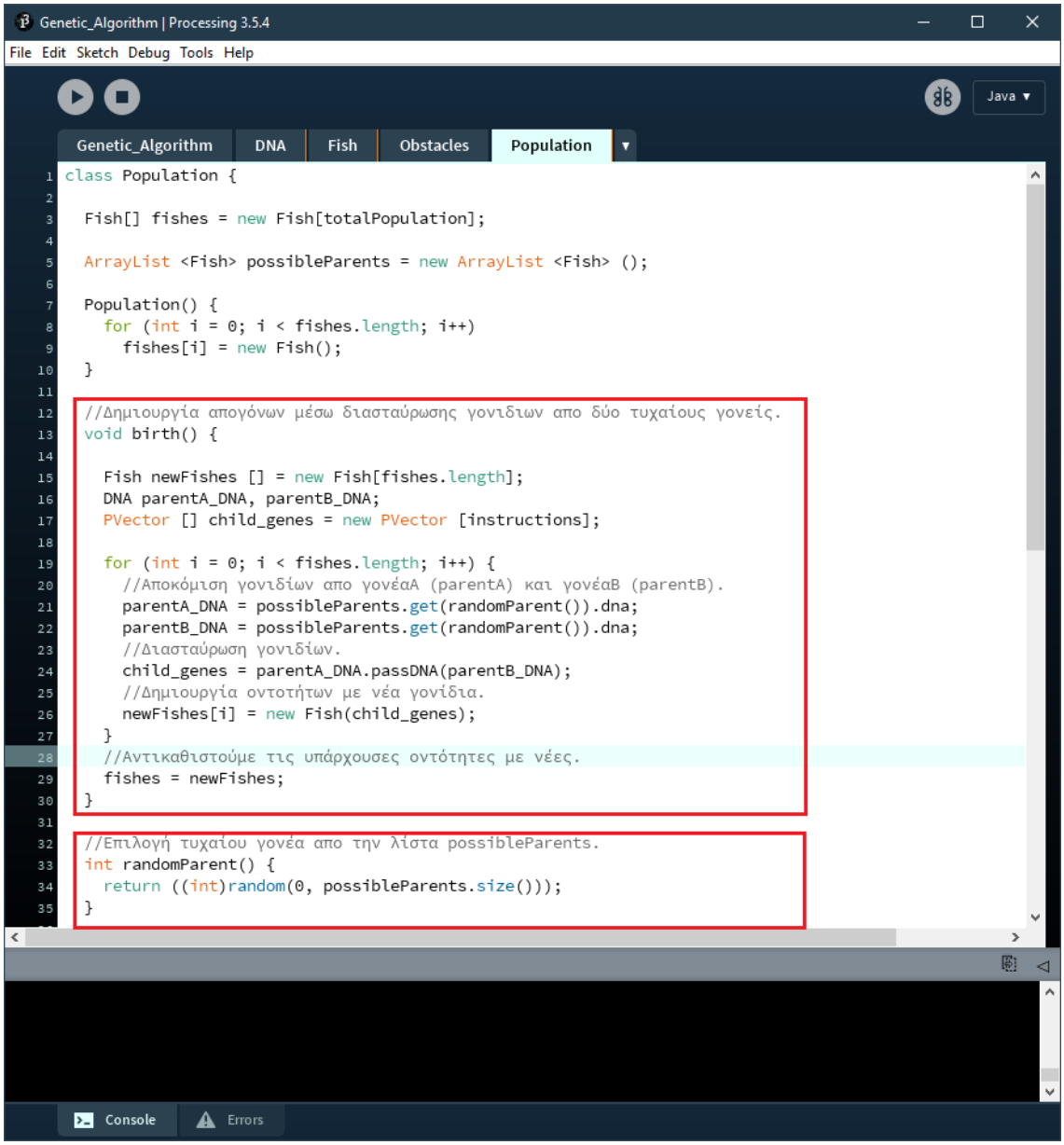
Έχοντας πλέον τα διασταυρωμένα γονίδια των δύο γονέων δημιουργούμε νέα οντότητα με τα υπάρχοντα γονίδια και την καταχωρούμε στον πίνακα newFishes. Αυτή η διαδικασία θα επαναληφθεί για όλες τις οντότητες του πληθυσμού.

Τέλος αντικαθιστούμε τις προηγούμενες οντότητες με τις νέες ώστε να μην αλλάξει η λειτουργικότητα του προγράμματος.

Τα αντικείμενα των γονέων επιλέγουν μία τυχαία οντότητα από την λίστα possibleParents με την βοήθεια της μεθόδου **randomParent().**

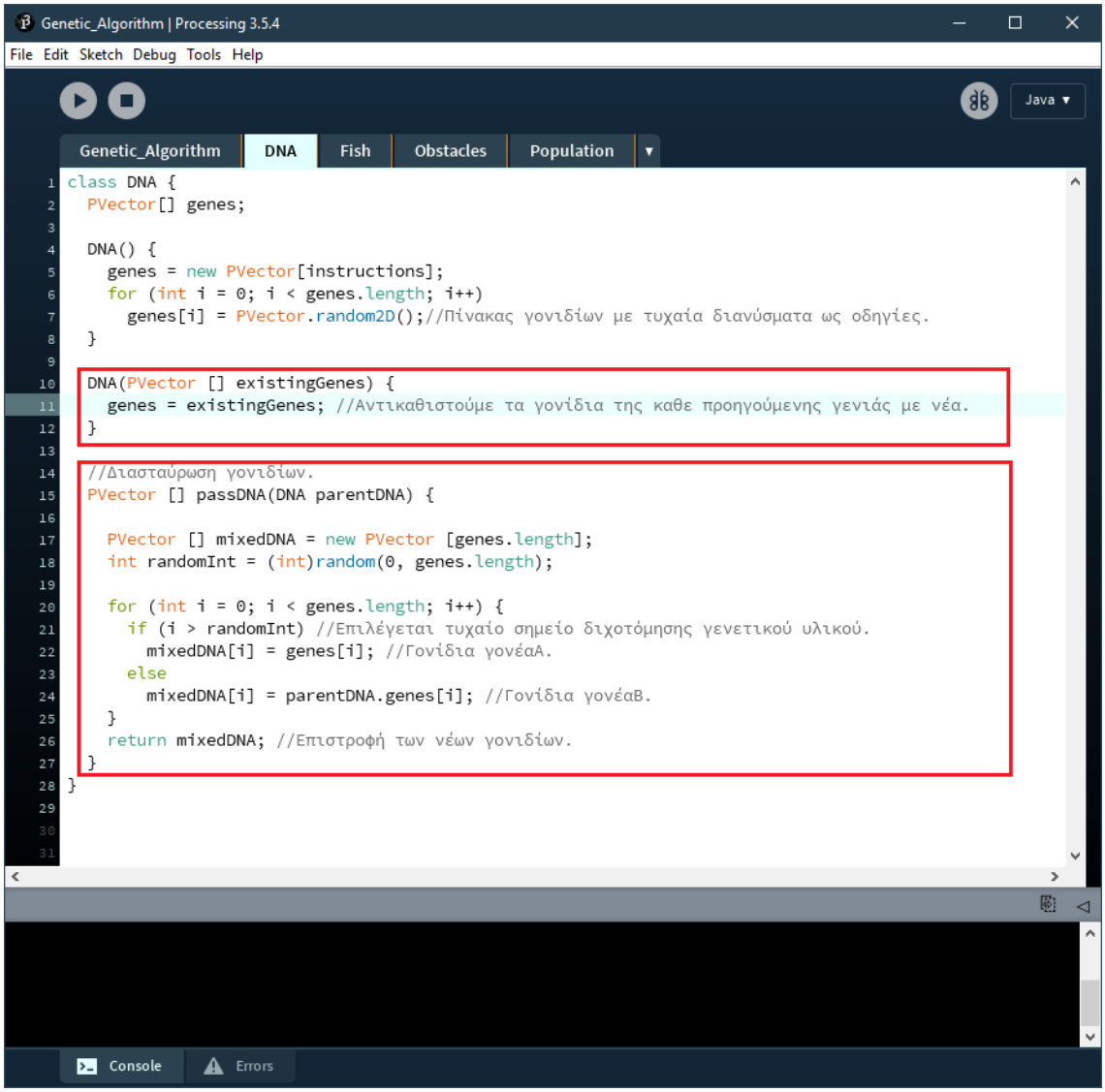
Επίσης ο πίνακας child_genes θα περιέχει τα δεδομένα (διανύσματα οδηγιών) των διασταυρωμένων γονιδίων προερχόμενα από τα αντικείμενα των γονέων.

Η διασταύρωση (crossover) των γονιδίων γίνεται με τη βοήθεια της μεθόδου **passDNA()** (Εικόνα 35).



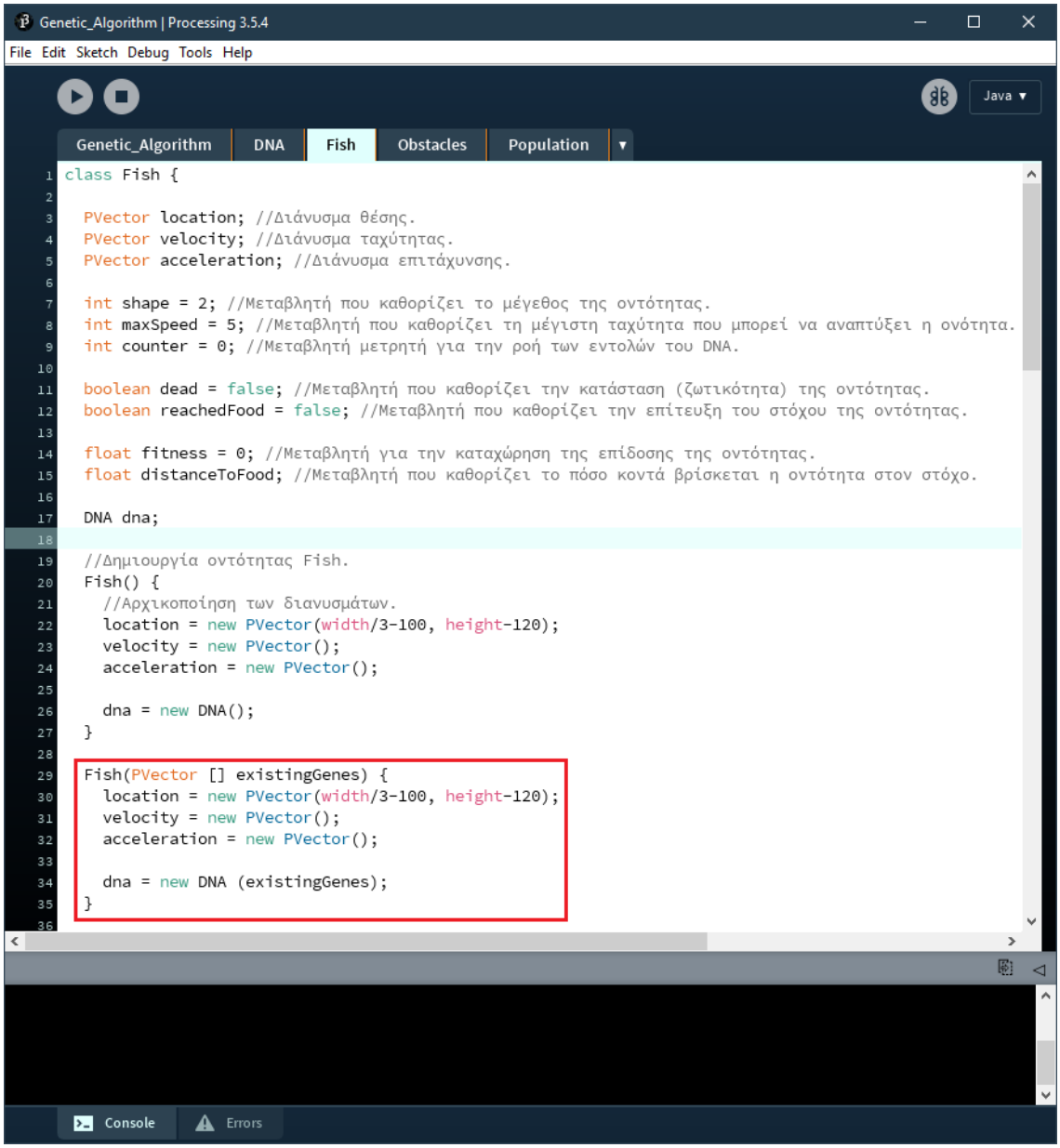
Εικόνα 34: Δημιουργία νέων οντοτήτων.

Η κλάση DNA έχει πλέον τη μέθοδο passDNA σκοπός της οποίας είναι η διασταύρωση των γονιδίων δύο γονέων και η επιστροφή τους σε μορφή πίνακα διανυσμάτων (Εικόνα 35). Δημιουργούμε λοιπόν τον πίνακα διανυσμάτων **mixedDNA** με μέγεθος ίδιο με τον πίνακα genes ώστε να περιέχει την αντίστοιχη ποσότητα οδηγιών. Ταυτόχρονα χρησιμοποιούμε μία νέα μεταβλητή την **randomInt**, η οποία επιλέγει έναν τυχαίο αριθμό από το μηδέν έως μία μονάδα μικρότερη από το μέγεθος των οδηγιών, αυτή η τιμή αποτελεί το σημείο διχοτόμησης του γενετικού υλικού των δύο γονέων. Στη συνέχεια καταχωρείται το πρώτο κομμάτι των γονιδίων του ενός γονέα και έπειτα το δεύτερο κομμάτι του άλλου γονέα. Έπειτα επιστρέφεται ο πίνακας των διασταυρωμένων γονιδίων.



Εικόνα 35: Μέθοδος διασταύρωσης γονιδίων (crossover).

Επιπλέον στη κλάση DNA υλοποιούμε έναν νέο κατασκευαστή ο οποίος θα δέχεται τα διασταυρωμένα γονίδια τα οποία θα αντικαθιστούν τα παλιά και θα εφαρμόζονται στη νέα γενιά οντοτήτων. Αυτή η νέα έκδοση του κατασκευαστή καλείται από μια επίσης νέα έκδοση του κατασκευαστή της Fish (Εικόνα 36).

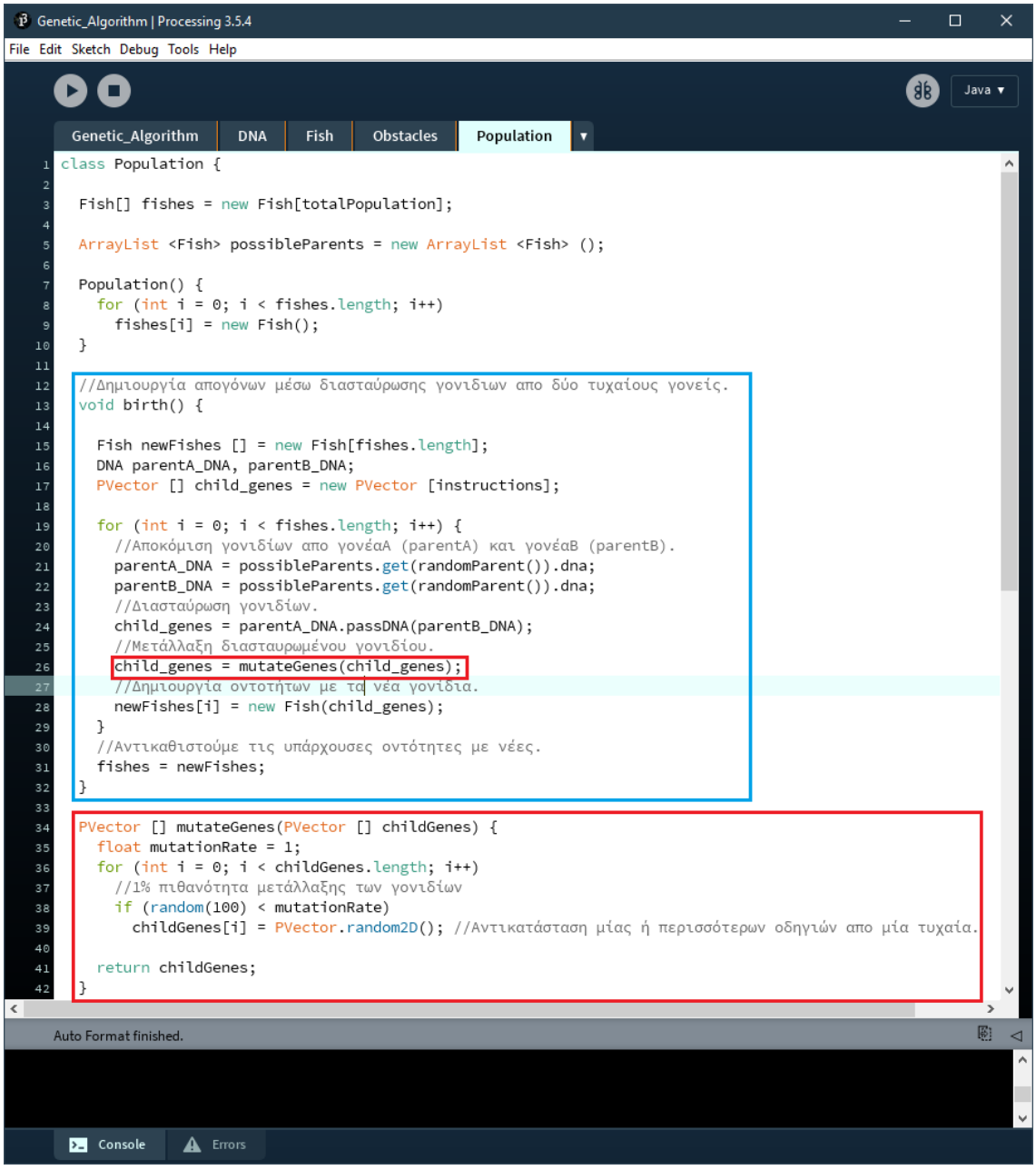


Εικόνα 36: Υλοποίηση δεύτερου Fish Constructor.

Πλέον είμαστε ένα βήμα πριν την εκκίνηση του γενετικού αλγορίθμου όμως σε αυτό το σημείο θα προσθέσουμε μία ακόμη λειτουργία, την μετάλλαξη των γονιδίων (Εικόνα 37).

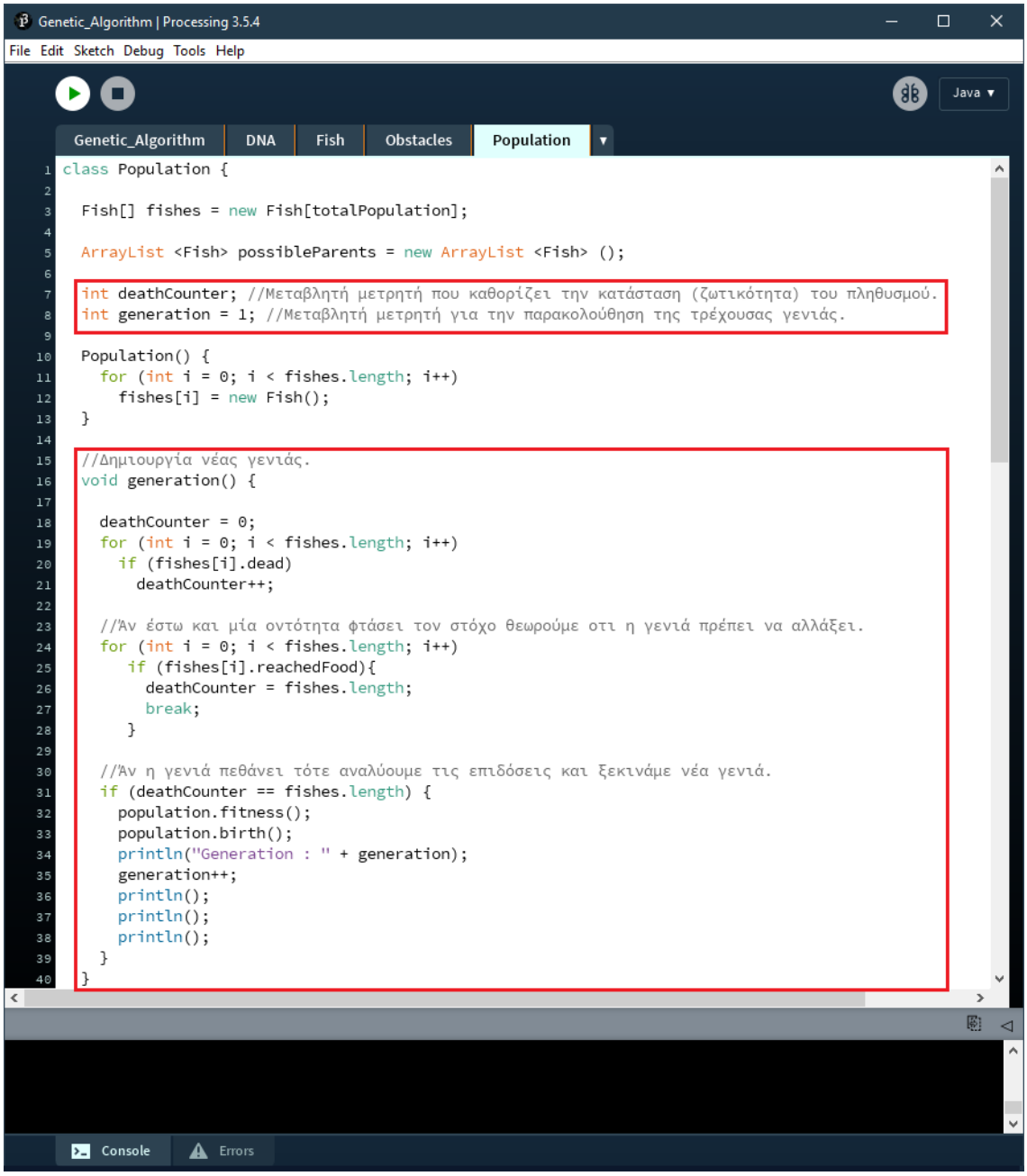
Άρα λοιπόν στην κλάση Population δημιουργούμε τη μέθοδο **mutateGenes()**. Στη μέθοδο διατρέχουμε ολόκληρο των πίνακα των διασταυρωμένων γονιδίων όπου για πιθανότητες της τάξης του ένα τοις εκατό (**1%**) ενδέχεται να αντικατασταθεί μία ή περισσότερες οδηγίες από μία τυχαία.

Η διαδικασία της μετάλλαξης γίνεται προτού δημιουργηθεί η νέα οντότητα και αφού έχει προηγηθεί η διασταύρωση των γονιδίων.



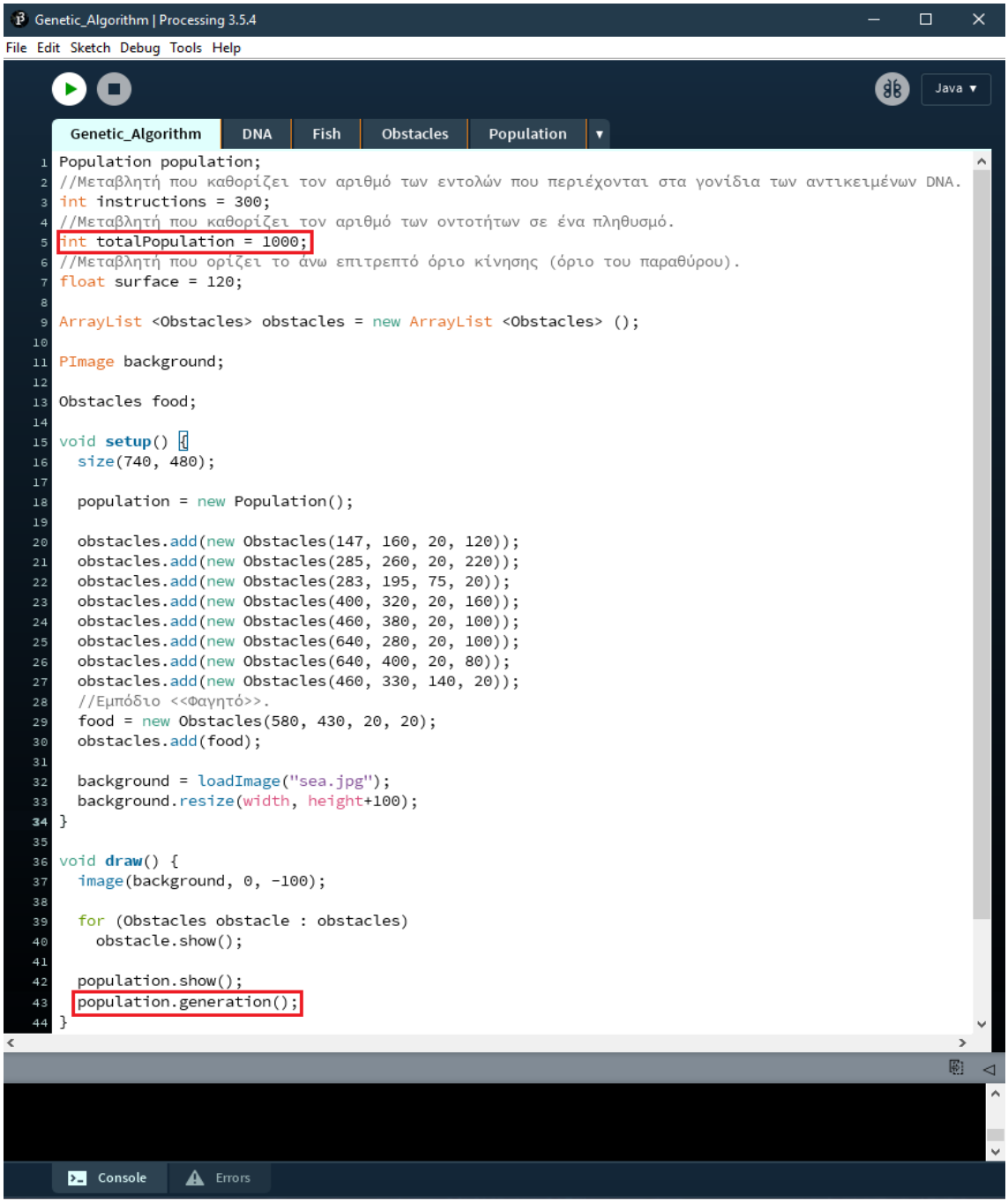
Εικόνα 37: Μέθοδος μετάλλαξης γονιδίων (mutation).

Φτάνοντας στο τέλος της υλοποίησης του γενετικού αλγορίθμου δημιουργούμε την μέθοδο **generation()** η οποία αναλαμβάνει την εκκίνηση της νέας γενιάς των οντοτήτων αφού πρώτα βεβαιωθεί ότι όλες οι οντότητες είναι νεκρές ή έστω και μία οντότητα έχει φτάσει στον στόχο (Εικόνα 38).



Εικόνα 38: Δημιουργία νέας γενιάς, υλοποίηση μεθόδου generation.

Ολοκληρώνοντας, στην κεντρική καρτέλα του προγράμματος καλούμε την μέθοδο generation και αλλάζουμε την ποσότητα του πληθυσμού σε 1000 (οντότητες) (Εικόνα 39).

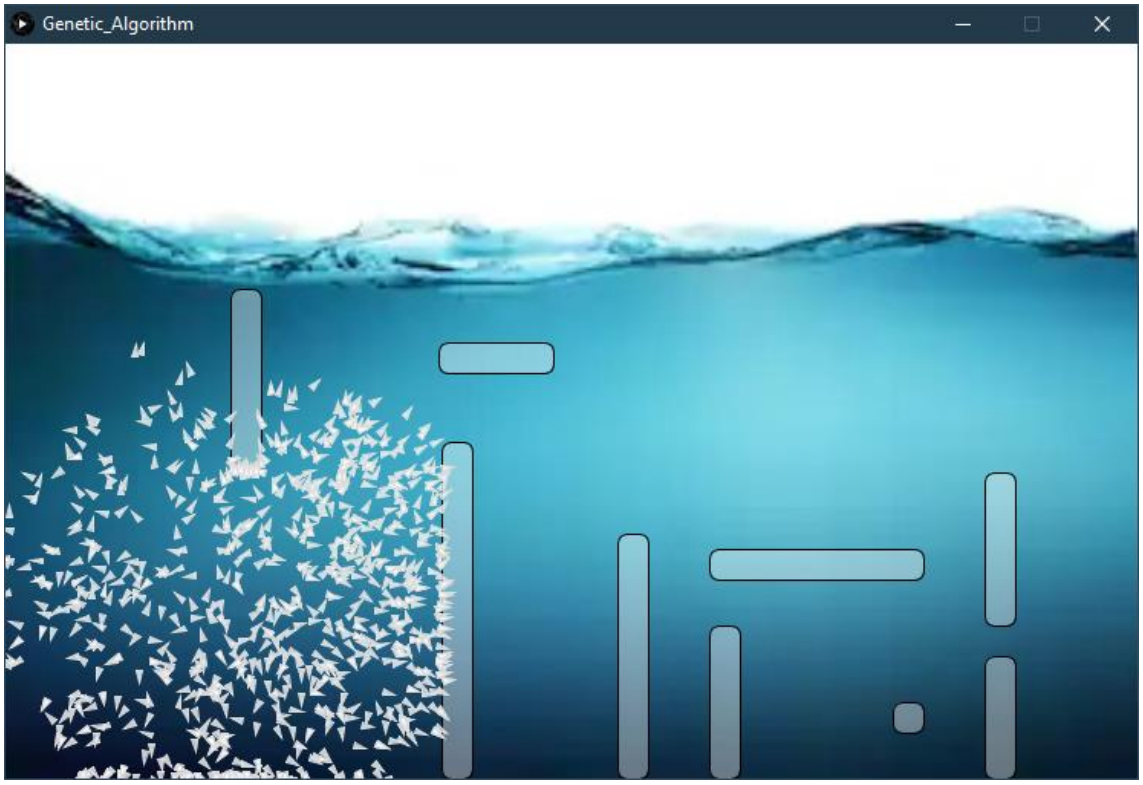



```
Population population;
//Μεταβλητή που καθορίζει τον αριθμό των εντολών που περιέχονται στα γονίδια των αντικειμένων DNA.
int instructions = 300;
//Μεταβλητή που καθορίζει τον αριθμό των οντοτήτων σε ένα πληθυσμό.
int totalPopulation = 1000;
//Μεταβλητή που ορίζει το άνω επιτρεπτό όριο κίνησης (όριο του παραθύρου).
float surface = 120;

ArrayList <Obstacles> obstacles = new ArrayList <Obstacles> ();

PImage background;

Obstacles food;

void setup() {
  size(740, 480);

  population = new Population();

  obstacles.add(new Obstacles(147, 160, 20, 120));
  obstacles.add(new Obstacles(285, 260, 20, 220));
  obstacles.add(new Obstacles(283, 195, 75, 20));
  obstacles.add(new Obstacles(400, 320, 20, 160));
  obstacles.add(new Obstacles(460, 380, 20, 100));
  obstacles.add(new Obstacles(640, 280, 20, 100));
  obstacles.add(new Obstacles(640, 400, 20, 80));
  obstacles.add(new Obstacles(460, 330, 140, 20));
  //Εμπόδιο <<Φαγητό>>.
  food = new Obstacles(580, 430, 20, 20);
  obstacles.add(food);

  background = loadImage("sea.jpg");
  background.resize(width, height+100);
}

void draw() {
  image(background, 0, -100);

  for (Obstacles obstacle : obstacles)
    obstacle.show();

  population.show();
  population.generation();
}
```

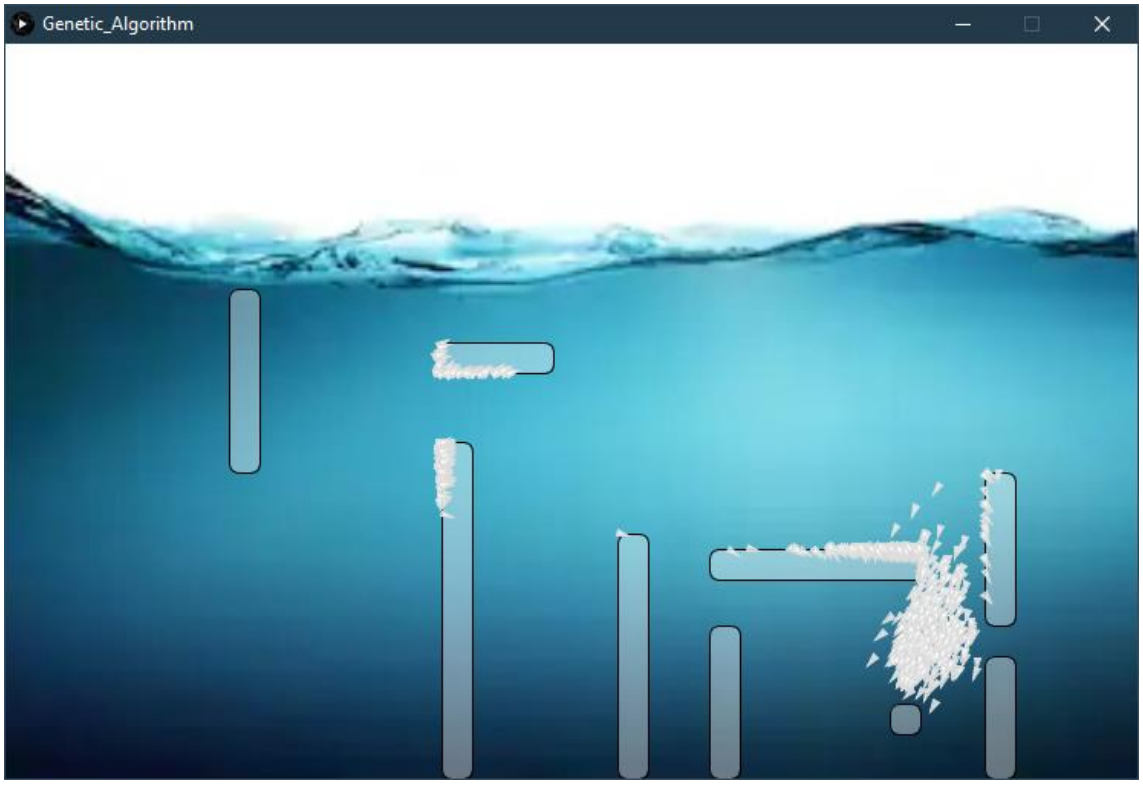
Εικόνα 39: Τελική μορφή κυρίου προγράμματος.

Παρακάτω στην εικόνα **40** οι οντότητες κινούνται τυχαία στον χώρο καθώς είναι η μηδενική γενιά και δεν έχει επέλθει ακόμα η εξέλιξη των γονιδίων τους.

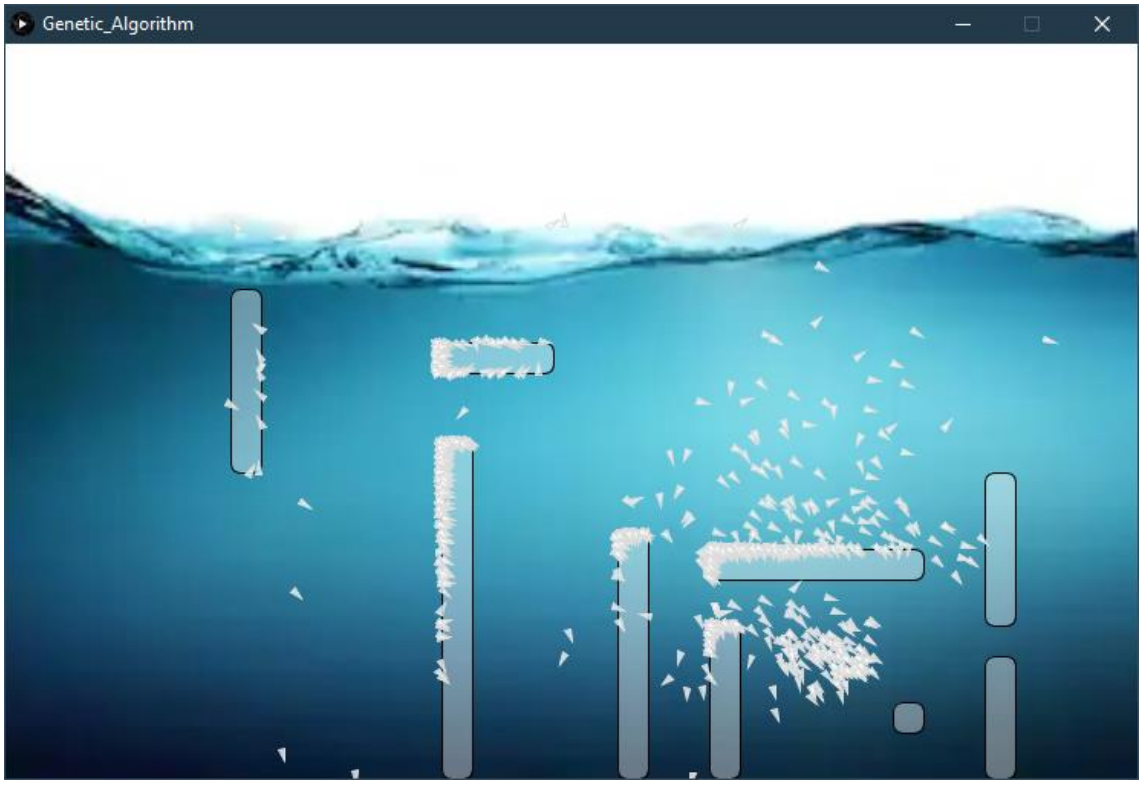
Στις εικόνες **41**,**42**,**43** παρατηρούμε τη συμπεριφορά των οντοτήτων μετά από την εξέλιξη αρκετών γενεών. Η εικόνα **41** απεικονίζει τις οντότητες οι οποίες έχουν επιλέξει μια μη βέλτιστη διαδρομή προς τον στόχο. Έπειτα στην εικόνα **42** μετά το πέρας αρκετών γενεών εξέλιξης οι οντότητες αρχίζουν να εξερευνούν τη βέλτιστη διαδρομή αυτή δηλαδή που απαιτεί λιγότερα βήματα από το σημείο αρχής μέχρι τον στόχο. Τέλος στην εικόνα **43** η πλειοψηφία των οντοτήτων εξελίσσεται στην ακολούθηση της βέλτιστης διαδρομής (λύσης) με τις λιγότερες δυνατές απώλειες.



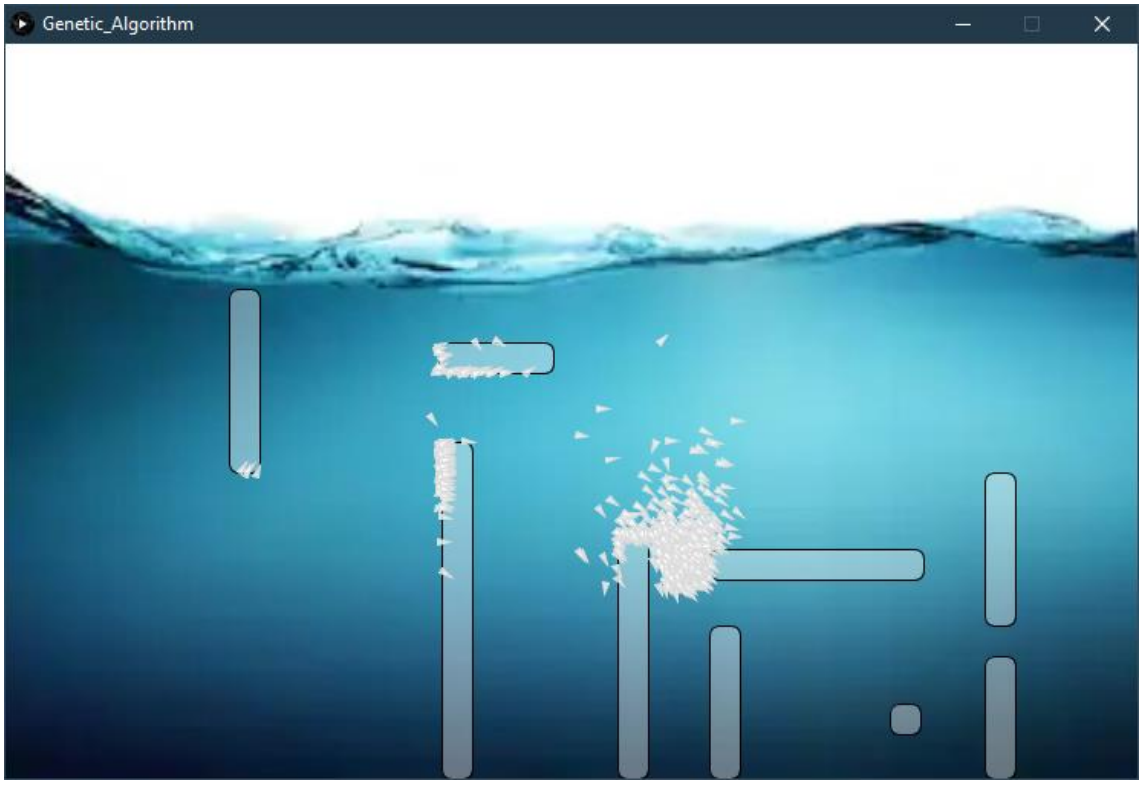
Εικόνα 40: Εκτέλεση κώδικα, γενιά 0.



Εικόνα 41: Εκτέλεση κώδικα, μη βέλτιστη διαδρομή.



Εικόνα 42: Εκτέλεση κώδικα, εξέλιξη στη βέλτιστη διαδρομή.



Εικόνα 43: Εκτέλεση κώδικα, βέλτιστη λύση της πλειοψηφίας.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Στη διπλωματική εργασία γνωρίσαμε το περιβάλλον ανάπτυξης της εφαρμογής (Processing) και δημιουργήσαμε στοιχειώδη παραδείγματα με σκοπό την εξοικείωση του αναγνώστη με την διεπαφή, την κατανόηση της συμπεριφοράς του εργαλείου καθώς και των ιδιαιτεροτήτων της γλώσσας.

Επιπλέον μοντελοποιήσαμε απλές προσομοιώσεις του φυσικού μας κόσμου που αποτελούν πρόλογο για την αντίληψη των βασικών ιδεών πάνω στις οποίες είναι βασισμένος ο κώδικας που δίνει τη λύση του προβλήματος που πραγματεύεται η εργασία. Αναλύθηκαν κάποιες βασικές αρχές της φυσικής και των μαθηματικών που κρίνονται απαραίτητες για γέννηση αυτών των συστημάτων και την αναπαράσταση των ιδιοτήτων τους. Επιπλέον εξετάσαμε την φύση των γενετικών αλγορίθμων με έμφαση στους μηχανισμούς τους οποίους επρόκειτο να εφαρμόσουμε στον κώδικα που εξετάζει τη λύση του προβλήματος.

Κατά την εκτέλεση του τελικού κώδικα παρατηρήσαμε ότι το σύστημα χρειάζεται εκπαίδευση προτού καταλήξει στη λύση. Μετά το πέρας μερικών γενεών εξέλιξης οι οντότητες μας βρίσκουν λύση στο πρόβλημα. Ακόμη κι αν δεν βρεθεί η βέλτιστη λύση μετά από αρκετή εκπαίδευση – εξέλιξη τότε κάποια στιγμή αναπόφευκτα θα βρεθεί η βέλτιστη η λύση. Φυσικά σημαντικός καθίσταται ο παράγοντας της τύχης, ο οποίος τις περισσότερες φορές καθορίζει αρκετά την έκβαση του αποτελέσματος.

Η εργασία μπορεί να θεωρηθεί και ως ένα μέσο εκμάθησης των λειτουργιών ενός γενετικού αλγορίθμου λόγω του ότι κάποιος μπορεί εύκολα να πειραματιστεί με τις παραμέτρους που ελέγχουν τη συμπεριφορά είτε των οντοτήτων είτε του περιβάλλοντος ακόμη και αυτές των μηχανισμών του γενετικού αλγορίθμου, δημιουργώντας έτσι μια διαδραστική αλληλεπίδραση του χρήστη με τον κώδικα.

Ακόμη η ύπαρξη της προσομοίωσης του οικοσυστήματος και των οντοτήτων επι των οποίων εφαρμόζεται ο αλγόριθμος προωθεί την άμεση αντίληψη της συμπεριφοράς ενός γενετικού αλγορίθμου.

Η χρήση των γενετικών αλγορίθμων είναι ευρεία και εφαρμόζεται σε ποικιλία προβλημάτων, στην εργασία έχει υλοποιηθεί ένας κλασσικός γενετικός αλγόριθμος, αυτό σημαίνει ότι κάποιος θα μπορούσε να χρησιμοποιήσει το ίδιο μοντέλο κάνοντας αναγκαίες παραλλαγές στις παραμέτρους του ώστε να προσαρμόσει τη λειτουργικότητά του σε κάποιο άλλο πρόβλημα όχι απαραίτητα παρεμφερές.

Συνολικά η υλοποίηση αποτέλεσε πρόκληση αφού έπρεπε να μελετηθούν πολύπλοκες έννοιες και στη συνέχεια να εφαρμοστούν σε μορφή κώδικα και να απεικονιστούν σε ένα γραφικό περιβάλλον.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Casey Reas, Ben Fry, John Maeda. "*Processing: A Programming Handbook for Visual Designers and Artists*". Massachusetts Institute of Technology, Cambridge: MIT Press, 2007.

Danny Kodicek. "*Mathematics and Physics for Programmers*". Hingham, Massachusetts, Charles River Media, 2005.

Karl Sims. "*Artificial Evolution for Computer Graphics*". Paper, ACM SIGGRAPH '91, Conference: Computer Graphics, Las Vegas, Nevada, 1991.