ΜΕΡΟΣ 1

Tutorial 1: Αλληλεπιδρώντας με το Greenfoot

Αυτό το tutorial εξηγεί τα βασικά στοιχεία της διεπαφής Greenfoot, και πως γίνεται η αλληλεπίδραση με το Greenfoot.

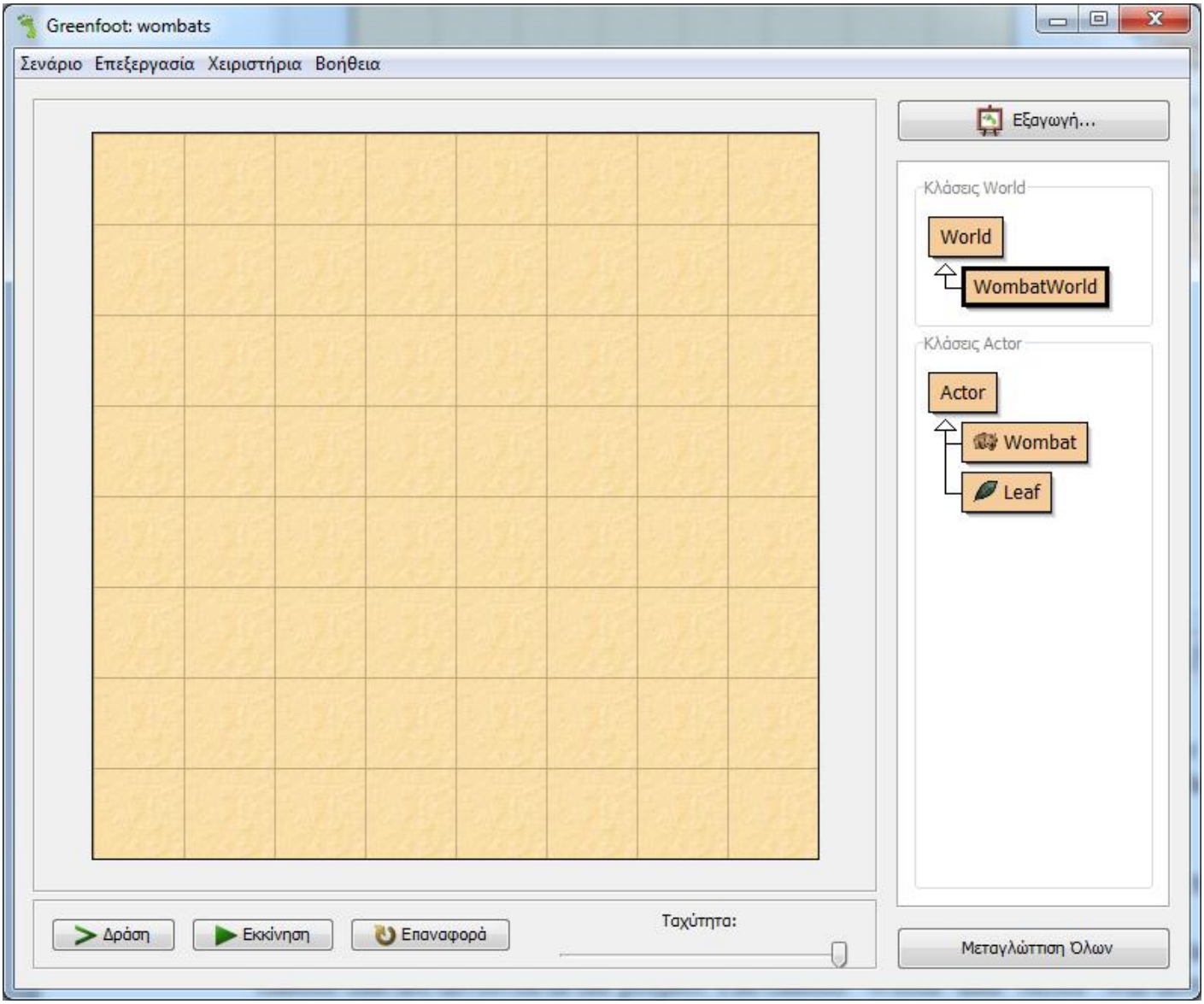
Αλλαγή διεπαφής στα Ελληνικά

Για να αλλάξουμε την διεπαφή του Greenfoot στα Ελληνικα ακολουθούμε τα εξής βήματα:

1. Εντοπίζουμε το αρχείο **greenfoot.defs.** Το αρχείο θα το βρείτε στο:
   <φάκελος_greenfoot> / lib / greenfoot.defs
2. Αυτό το αρχείο περιέχει παραμέτρους της εφαρμογής στην μορφή:
   όνομα_παραμέτρου = τιμή
3. Εντοπίζουμε την παράμετρο στο αρχείο που ονομάζεται:
   bluej.language
4. Σε αυτήν την παράμετρο θέτουμε την γλώσσα στην οποία επιθυμούμε να εμφανίζεται η διεπαφή μας. Για τα Ελληνικά δώστε το εξής:
   bluej.language = greek
5. Αυτό ήταν όλο. Έχετε κατά νου ότι η αλλαγές που κάναμε θα μεταφράσουν την διεπαφή στα Ελληνικά αλλά όχι και τον κώδικα των σεναρίων που εκτελείται ή τα ονόματα των κλάσεων.

Η διεπαφή Greenfoot

Το tutorial αυτό χρησιμοποιεί ένα σενάριο που ονομάζεται ‘wombats’ το οποίο διανέμεται με το Greenfoot.
Ανοίξτε το σενάριο wombats στο Greenfoot. Πρέπει να εμφανίζεται αυτό:



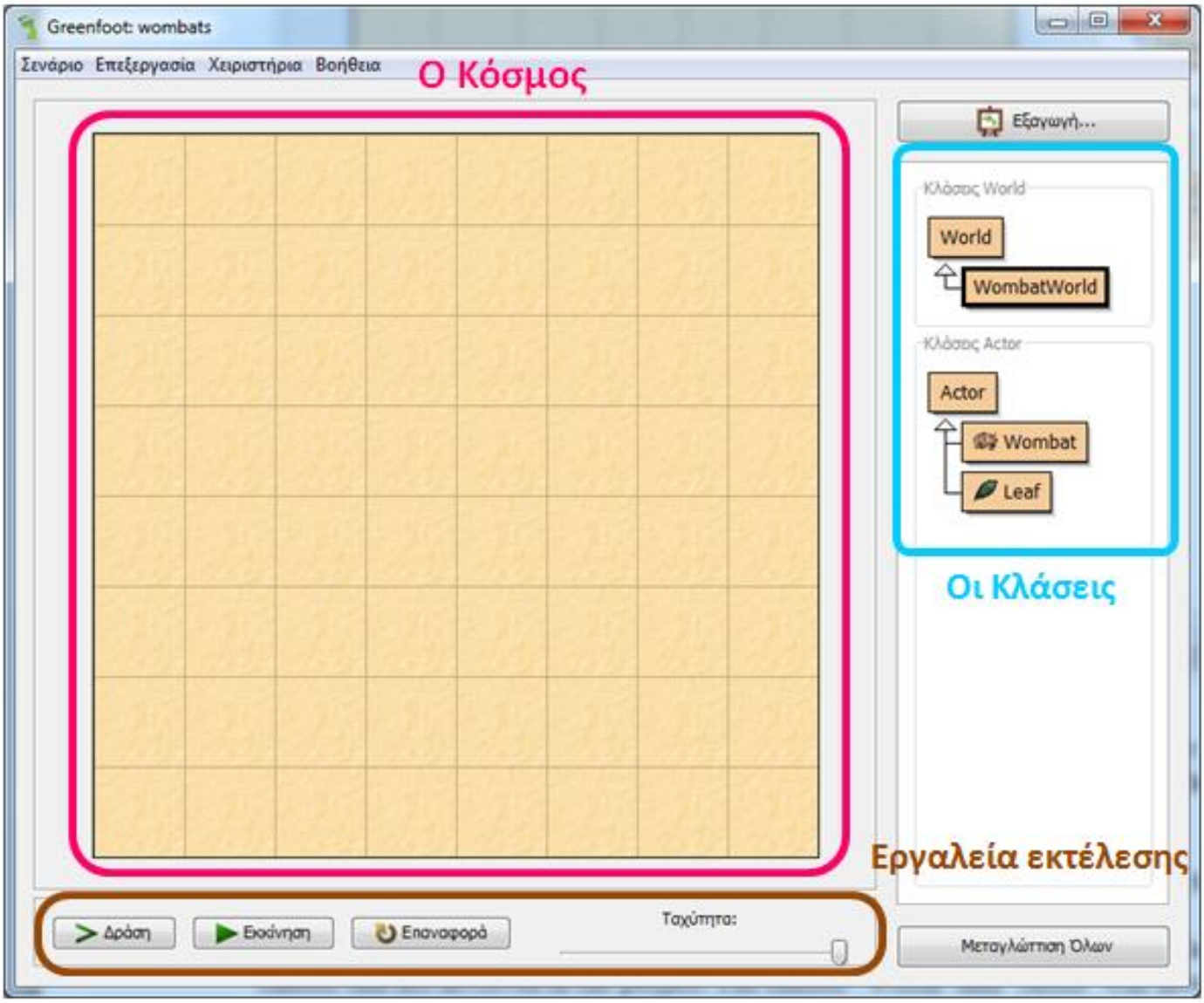
Αν δεν βλέπετε το world και οι κλάσεις στα δεξιά έχουν διαγώνιους καθέτους πάνω τους, είναι επειδή ο κώδικας δεν είναι μεταγλωττισμένος (compiled). Πατήστε στο κουμπί ‘Μεταγλώττιση Όλων’ κάτω δεξιά.

Η μεγάλη περιοχή με το πλέγμα που καλύπτει την πλειοψηφία του παραθύρου, ονομάζεται ‘the world’ δηλαδή ‘ο κόσμος’ της εφαρμογής μας.

Εφόσον έχουμε ένα σενάριο το οποίο έχει να κάνει με wombats βλέπουμε ένα wombat world.

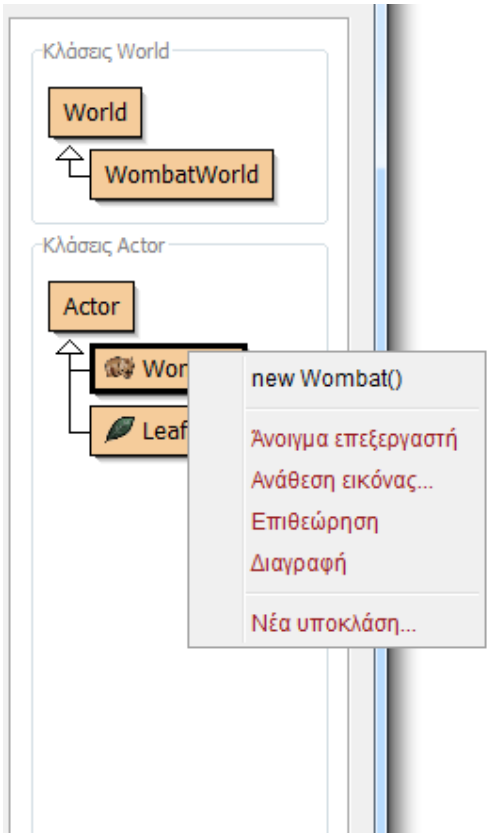
Στην δεξιά μεριά του παραθύρου είναι η οθόνη των κλάσεων. Εδώ μπορούμε να δούμε όλες τις java κλάσεις που εμπεριέχονται στο σενάριο (project). Οι κλάσεις ‘world’ και ‘actor’ θα είναι πάντα εκεί — Είναι από το σύστημα του Greenfoot. Οι υπόλοιπες κλάσεις ανήκουν στο σενάριο wombat και θα είναι διαφορετικές ανάλογα με το σενάριο που ανοίγετε ή δημιουργείτε. Κάτω από τον ‘Κόσμο’ είναι τα εργαλεία εκτέλεσης (Η περιοχή με τα κουμπιά ‘Δράση’, ‘Εκκίνηση’ και ‘Επαναφορά’ και τον ρυθμιστή της ταχυτητας).

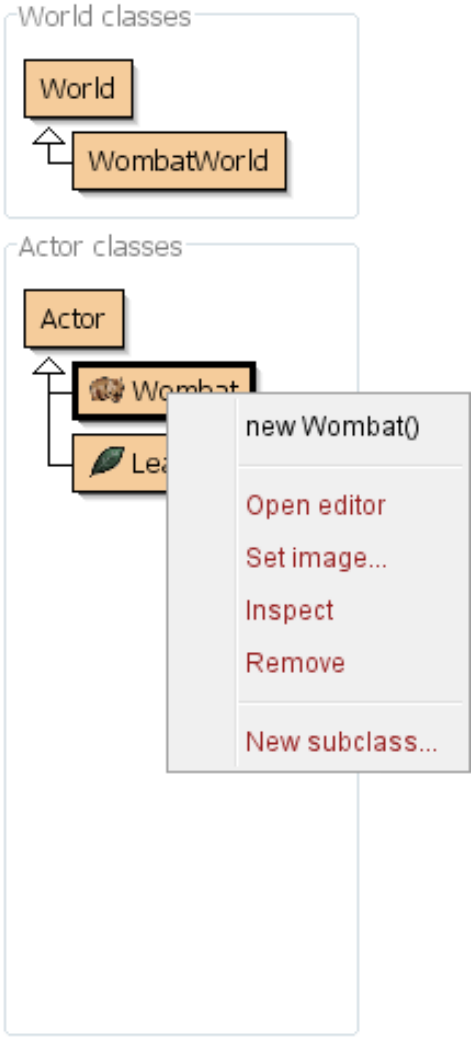
Ας επισημάνουμε όλα τα παραπάνω στην διεπαφή μας:



Τοποθέτηση αντικειμένων στο World

Τώρα θα τοποθετήσουμε κάποια αντικείμενα μέσα στο World. Δεξί κλικ (Στα Mac, Control-κλικ όποτε αναφέρεται στο tutorial αυτό δεξί κλικ) στην Wombat κλάση στην οθόνη των κλάσεων. Θα δείτε ένα αναδυόμενο μενού σαν το παρακάτω:


Κλάσεις World
World
WombatWorld
Κλάσεις Actor
Actor
Wor
Leaf
new Wombat()
Άνοιγμα επεξεργαστή
Ανάθεση εικόνας...
Επιθεώρηση
Διαγραφή
Νέα υποκλάση...


World classes
World
WombatWorld
Actor classes
Actor
Wombat
Lea
new Wombat()
Open editor
Set image...
Inspect
Remove
New subclass...

Διαλέξτε 'new Wombat()' απ' το μενού. Μετά κάντε κλικ οπουδήποτε μέσα στον Κόσμο. Μόλις δημιουργήσατε ένα 'Wombat' (σε όρους Java: ένα αντικείμενο) και το τοποθετήσατε μέσα στον 'Κόσμο'.

Τα Wombats τρώνε τα 'Leaf' (δηλαδή, τα φύλλα), οπότε ας βάλουμε μερικά στον Κόσμο. Κάντε δεξί κλικ στην κλάση Leaf, επιλέξτε 'new Leaf()' και τοποθετήστε το μέσα στον Κόσμο.

Υπάρχει και ένας άλλος τρόπος για την τοποθέτηση διαφόρων αντικειμένων λίγο πιο γρήγορα. Σιγουρευτείτε πως η κλάση Leaf είναι επιλεγμένη (Αριστερό κλικ πάνω στον πίνακα κλάσεων, και θα πάρει ένα πιο παχύ μαύρο περίγραμμα), στη συνέχεια, κρατήστε πατημένο το πλήκτρο Shift και πατήστε το αριστερό κλικ στον Κόσμου αρκετές φορές. Θα τοποθετηθεί ένα αντικείμενο από την επιλεγμένη κλάση σε κάθε κλικ. Πολύ πιο γρήγορα!

Κάντε τα αντικείμενα ενεργά

Πατήστε το κουμπί 'Δράση' στα εργαλεία εκτέλεσης. Κάθε αντικείμενο τώρα είναι ενεργό – που σημαίνει: κάθε αντικείμενο κάνει ό, τι έχει προγραμματιστεί να κάνει. Στο παράδειγμά μας, τα φύλλα δεν έχουν προγραμματιστεί να κάνουν τίποτα, ενώ τα wombats είναι προγραμματισμένα να προχωρούν. Δοκιμάστε να τοποθετήσετε δύο wombats στον κόσμο και πατήστε πάλι Act. Και τα δύο θα κινηθούν.

Στα Wombats αρέσει να τρώνε φύλλα. Αν τύχει και βρουν ένα φύλλο στο δρόμο τους, θα το φάνε. Προσπαθήστε να τοποθετήσετε μερικά φύλλα μπροστά στο Wombat και πατήστε 'Δράση' – το wombat θα προχωρήσει μπροστά και θα φάει τα φύλλα.

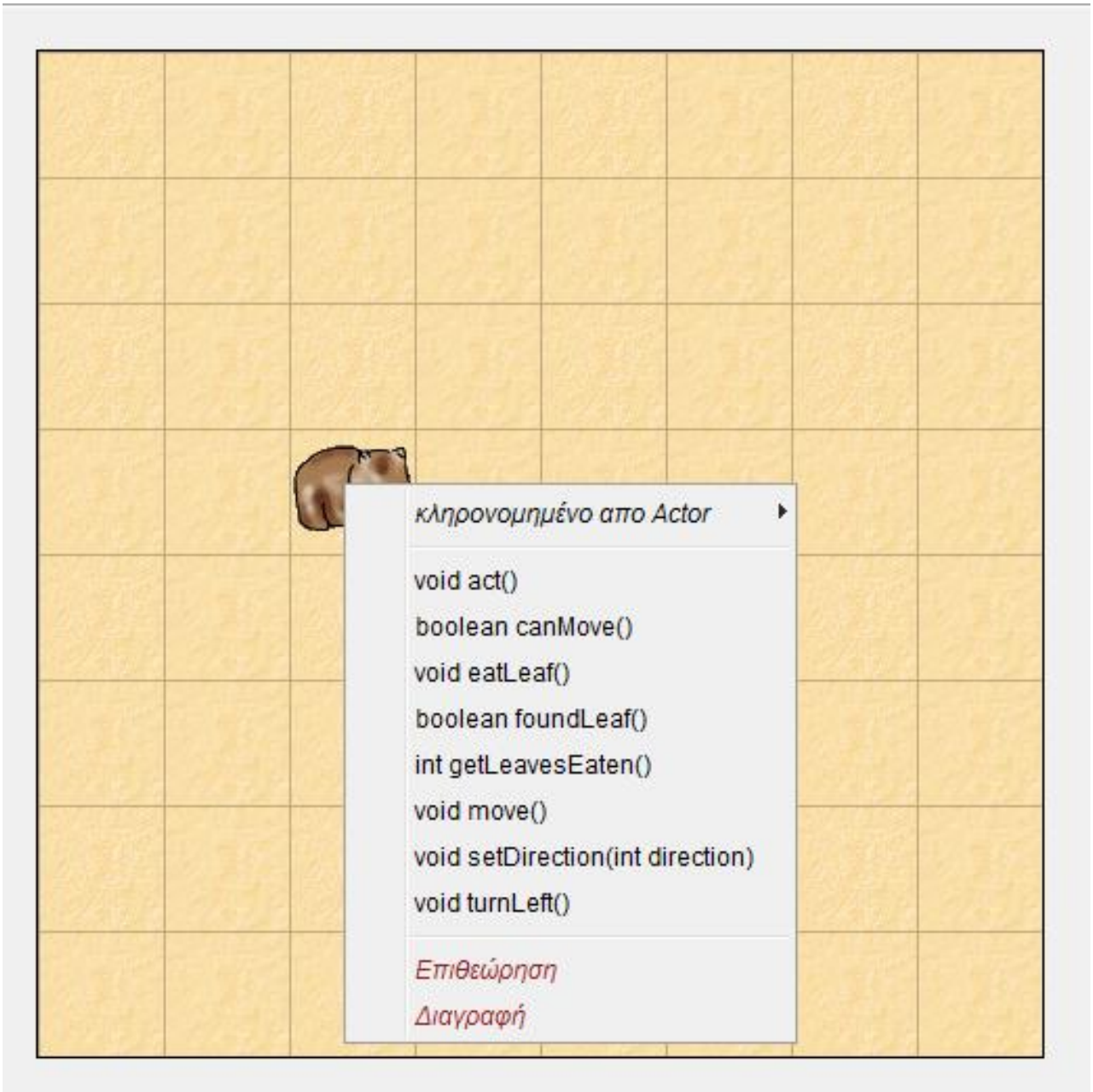
Εκτελέστε ένα σενάριο

Κάντε κλικ στο κουμπί 'Εκτέλεση'. Αυτό ισοδυναμεί με το να κάνετε κλικ στο κουμπί 'Δραση' ξανά και ξανά, πολύ γρήγορα. Θα παρατηρήσετε ότι το κουμπί 'Εκτέλεση' αλλάζει σε κουμπί 'Παύση'. Κάνοντας κλικ στο 'Παύση' σταματούν τα πάντα να ενεργούν.
Ο ρυθμιστής δίπλα στα κουμπιά 'Εκτέλεση' και 'Δράση' ρυθμίζει την ταχύτητα. Κάντε κλικ στην επιλογή 'Εκτέλεση' και στη συνέχεια ρυθμίστε τον Slider, και θα δείτε τη διαφορά.

Κάντε κλήση στις μεθόδους άμεσα

Αντί απλά να τρέχει όλο το σενάριο, μπορείτε επίσης να επικαλεστείτε συγκεκριμένες μεθόδους. Μια μέθοδος είναι μια μοναδική πράξη που ένα αντικείμενο μπορεί να εκτελέσει.

Βεβαιωθείτε ότι έχετε μια Wombat στον κόσμο, και το σενάριο δεν τρέχει. Στη συνέχεια, κάντε δεξί κλικ στο Wombat (το Wombat που υπάρχει στον κόσμο, όχι στην Wombat κλάση), και θα δείτε ότι τα αντικείμενα στον κόσμο έχουν επίσης ένα αναδυόμενο μενού:



Μπορείτε να επιλέξετε κάποια από τις μεθόδους που αναφέρονται εδώ για να ζητήσετε στο Wombat να κάνει κάτι.
Δοκιμάστε, για παράδειγμα το 'turnLeft()'. Επιλέγοντας αυτό από το μενού, λέτε στο Wombat να γυρίσει προς τα αριστερά. Δοκιμάστε επίσης και το 'move()'.

Μερικές μέθοδοι δίνουν μια απάντηση. Το 'getLeavesEaten()', για παράδειγμα, θα σας πει πόσα φύλλα έχει φάει μέχρι τώρα το Wombat. Δοκιμάστε το. Στη συνέχεια, βάλτε το Wombat να φάει ένα ακόμα φύλλο, και προσπαθήστε να κάνετε κλήση ξανά αυτής της μεθόδου.

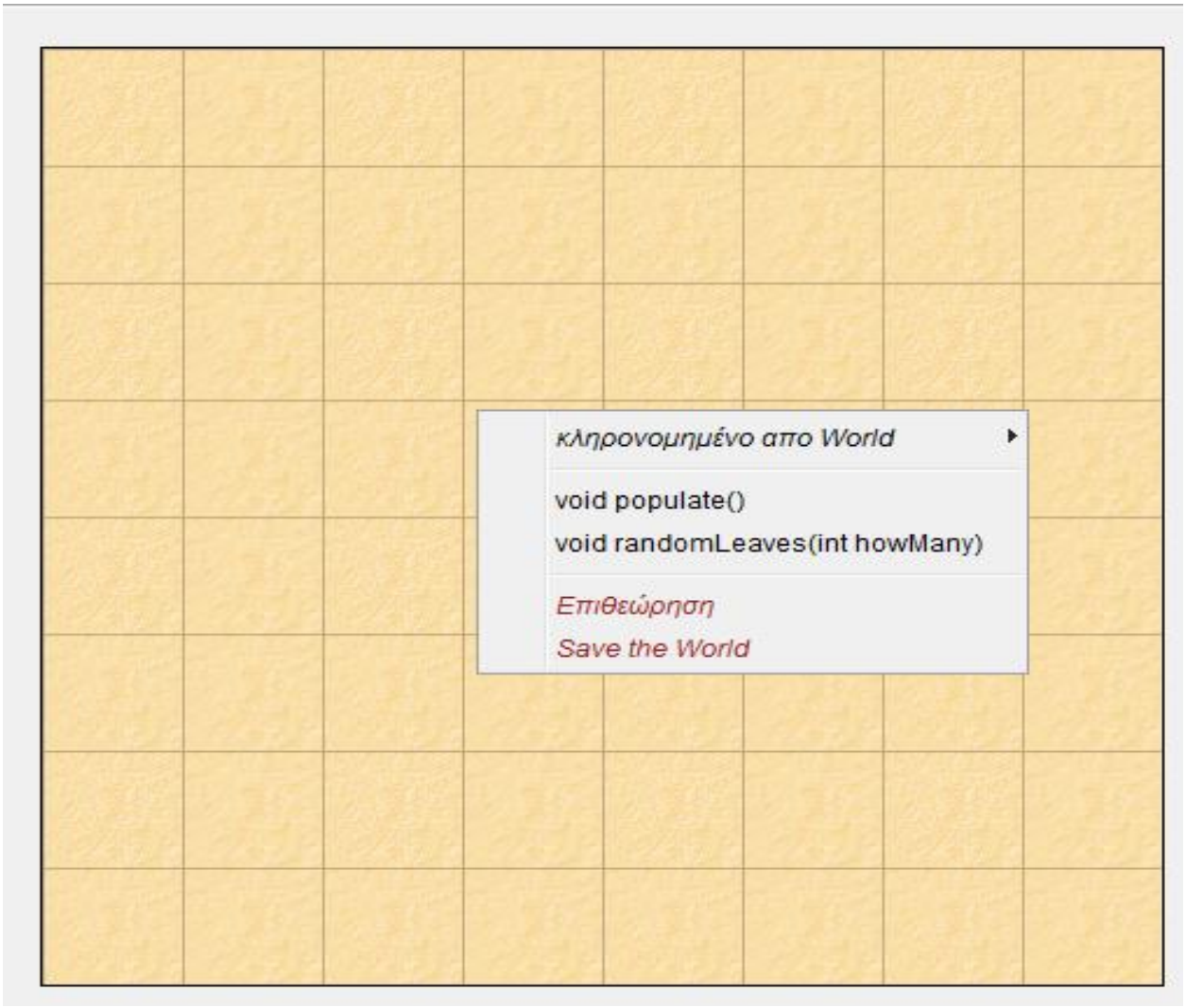
Θα δείτε επίσης μια μέθοδο που ονομάζεται 'act()'. Αυτή η μέθοδος καλείται κάθε φορά που κάνετε κλικ στο κουμπί 'Δράση'. Αν θέλετε ένα αντικείμενο μόνο να ενεργήσει αντί του συνόλου των αντικειμένων στον κόσμο, μπορείτε να το κάνετε, με την κλήση της μεθόδου 'act()' του συγκεκριμένου αντικειμένου που θέλετε να ενεργήσει.

Δημιουργία ενός καινούριου Κόσμου

Εάν έχετε πολλά αντικείμενα στον κόσμο τα οποία δεν θέλετε πια, και θέλετε να ξεκινήσετε από την αρχή, υπάρχει μια εύκολη λύση: Μπορείτε να πετάξετε τον κόσμο και δημιουργήσετε ένα καινούριο. Αυτό γίνεται συνήθως, κάνοντας κλικ στο κουμπί 'Επαναφορά' από τα εργαλεία εκτέλεσης. Θα έχετε ένα νέο, άδειο κόσμο. Ο παλιός κόσμος διαγράφεται (και μαζί με αυτόν όλα τα αντικείμενα που υπήρχαν μέσα στον κόσμο) - μπορείτε να έχετε μόνο έναν κόσμο κάθε φορά..

Καλέστε την μέθοδο Κόσμος

Έχουμε δει ότι τα αντικείμενα στον κόσμο έχουν μεθόδους που μπορείτε να επικαλεστείτε μέσω ενός αναδυόμενου μενού. Ο ίδιος ο κόσμος είναι επίσης είναι ένα αντικείμενο που έχει μεθόδους τις οποίες μπορείτε να επικαλεστείτε. Κάντε δεξί κλικ σε οποιοδήποτε κενό χώρο στον κόσμο, ή στην γκρίζα περιοχή ακριβώς δίπλα στον κόσμο, και θα δείτε το μενού του κόσμου:



Μία από τις μεθόδους αυτού του μενού είναι η 'populate()'. Δοκιμάστε την. Είναι μια μέθοδος η οποία δημιουργεί πολλά φύλλα και Wombats και τα τοποθετεί μέσα στον κόσμο. Μπορείτε να εκτελέσετε στη συνέχεια το σενάριο.

Μια άλλη μέθοδος Κόσμου είναι η 'randomleaves(int howMany)'. Αυτή η μέθοδος τοποθετεί μερικά φύλλα στον κόσμο σε τυχαίες θέσεις. Σημειώστε ότι αυτή η μέθοδος έχει μερικές λέξεις μεταξύ των παρενθέσεων'int howMany'. Αυτό ονομάζεται 'παράμετρος' και σημαίνει ότι θα πρέπει να ορίσετε κάποιες επιπλέον πληροφορίες, όταν επικαλείστε αυτή τη μέθοδο. Ο όρος 'int' σας λέει ότι ένας ακέραιος αριθμός αναμένεται, και το ''HowMany'' υποδηλώνει πως θα πρέπει να προσδιορίσετε πόσα φύλλα

θέλετε όταν επικαλεστείτε τη συγκεκριμένη μέθοδο. Ένας διάλογος θα εμφανιστεί που σας επιτρέπει να εισαγάγετε μια τιμή για αυτήν την παράμετρο. Πληκτρολογήστε έναν αριθμό (π.χ.: 12) και πατήστε Ok.

(Μπορείτε να παρατηρήσετε, αν μετρήσετε, ότι μερικές φορές έχουν δημιουργηθεί λιγότερα φύλλα από τον καθορισμένο αριθμό. Αυτό συμβαίνει επειδή μερικά φύλλα μπορεί να είναι στην ίδια θέση, και βρίσκεται το ένα πάνω από το άλλο.)

Αυτά λοιπόν είναι αρκετά για το τρέξιμο των wombats σε ατελείωτους κύκλους - ας περάσουμε στην πραγματικά ενδιαφέροντα πράγματα: Τον προγραμματισμό! Αυτό είναι στο μέρος 2 του tutorial.

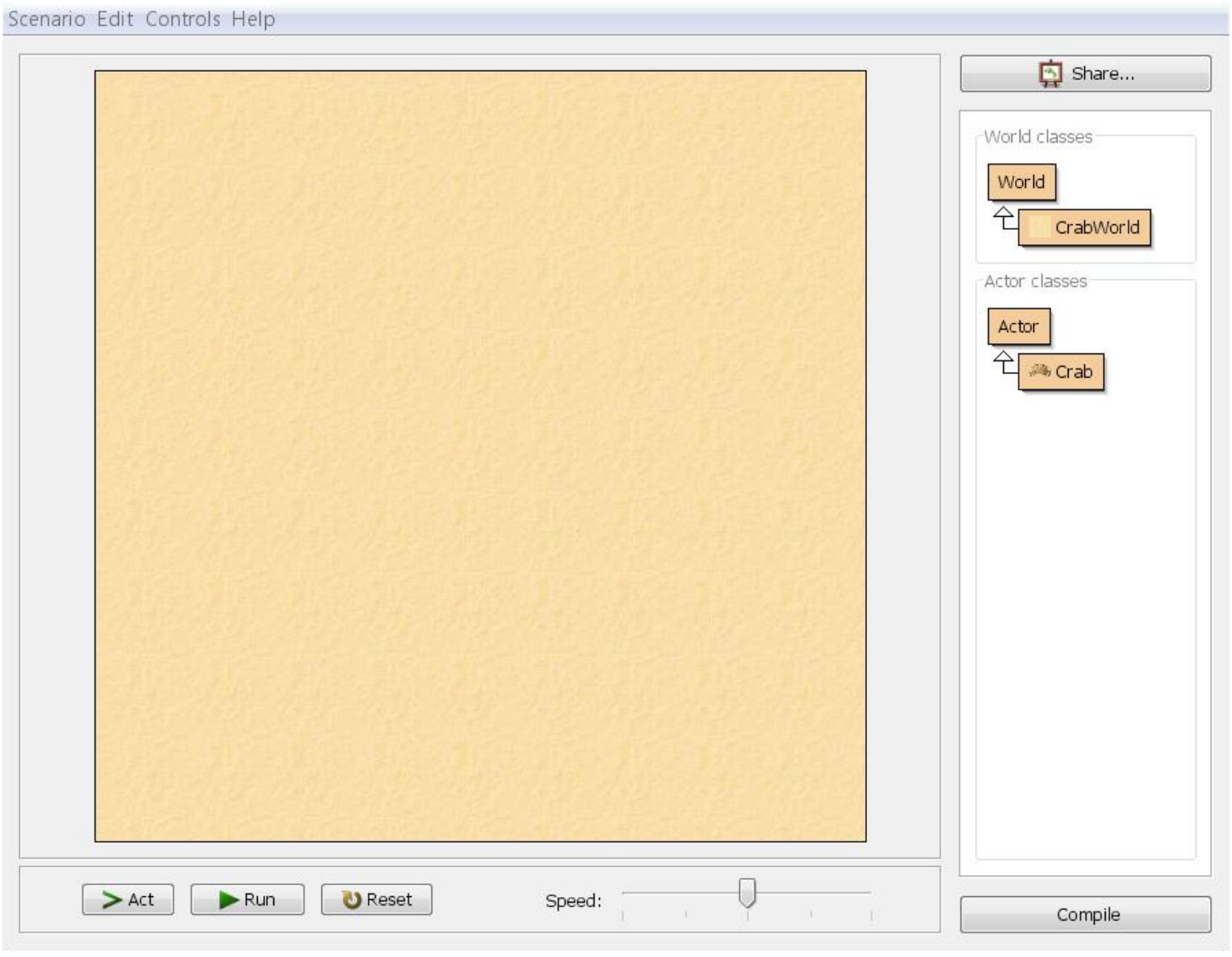
ΜΕΡΟΣ 2

Tutorial 2:Κίνηση και Κλειδιά ελέγχου

Αυτό το tutorial θα εξηγήσει πώς να κάνουμε κίνηση στο Greenfoot, και πώς να την ελέγχουμε με το πληκτρολόγιο.

The Crabs Scenario

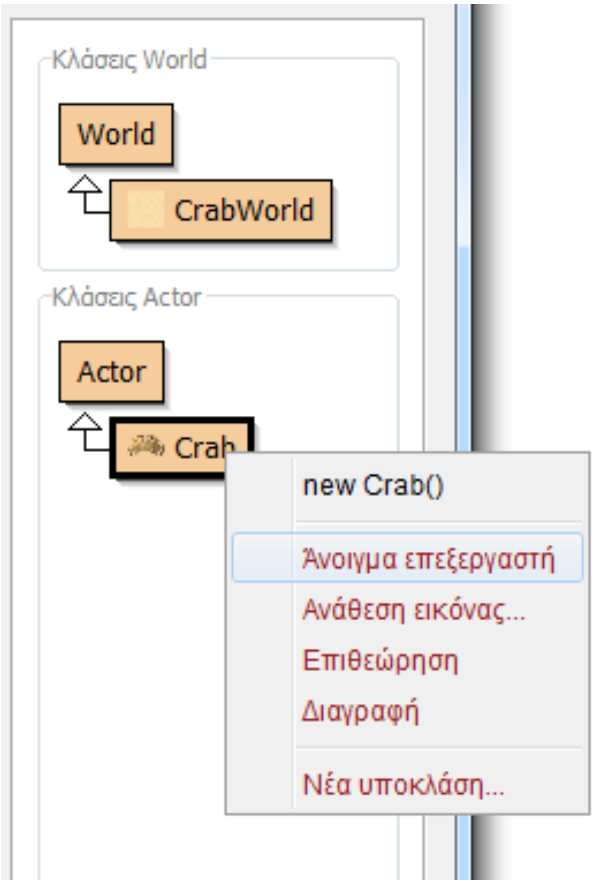
Κατεβάστε το crab zip file ένα σενάριο με καβούρια και αποσυμπιέστε τα περιεχόμενα του κάπου στον σκληρό σας δίσκο.
Στη συνέχεια, ανοίξτε το σενάριο από την περιοχή που το αποθηκεύσατε στο Greenfoot. Θα πρέπει να δείτε στην διεπαφή του Greenfoot, ένα κενό αμμώδη κόσμο:



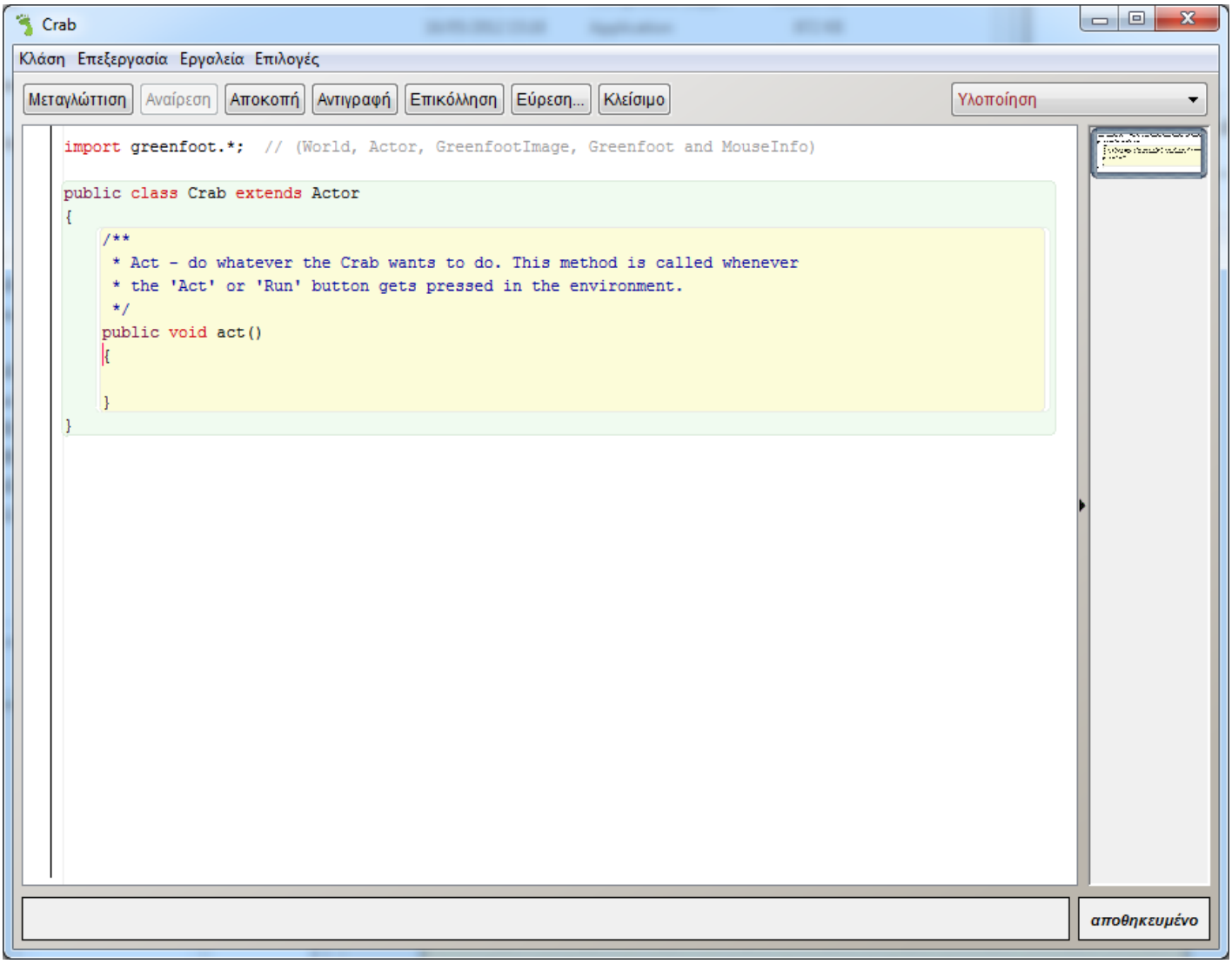
Κάντε δεξί κλικ πάνω στην κλάση 'Crab' (δηλαδή, καβούρι) και επιλέξτε 'new Crab()'. Κάντε κλικ τον κόσμο για να τοποθετήσετε το καβούρι:

Μετά από αυτό, κάντε κλικ στην επιλογή Run. Μπορεί να ελπίζετε πως θα δείτε καβούρι να κάνει ένα εκπληκτικό χορό γύρω από την οθόνη. Δυστυχώς όμως, φαίνεται ότι έχουμε ένα τεμπέλικο Καβούρι!

Ας ανοίξουμε τον κώδικα και να ρίξουμε μια ματιά. Μπορείτε είτε να κάνετε διπλό κλικ στην κλάση καβουριών στο πρόγραμμα περιήγησης κλάσεων, ή μπορείτε να κάνετε δεξί κλικ και να επιλέξτε "Άνοιγμα Επεξεργαστή:



Αυτό που θα δείτε είναι ο κώδικας Java για τον Κάβουρα. Δεν χρειάζεται ακόμα να τα κατανοήσετε όλα αυτά ακριβώς , αλλά το σημαντικό κομμάτι είναι ο κώδικας ανάμεσα στις αγκύλες κάτω από το public void act()- τώρα δεν υπάρχει τίποτα μέσα



Μια γρήγορη σημείωση σχετικά με το τι είναι αυτές οι παρενθέσεις (ως συνήθως, η Wikipedia έχει ένα τρομακτικά περιεκτικό άρθρο το οποίο όμως είναι στα Αγγλικά http://en.wikipedia.org/wiki/Brackets). Υπάρχουν τρεις βασικοί τύποι παρενθέσεων που χρησιμοποιούνται στην Java; Ας τους δούμε:

( )

Αυτά είναι στρογγυλές παρενθέσεις, γνωστά στις ΗΠΑ ως parentheses, αλλά στο Ηνωμένο Βασίλειο απλώς ως brackets. Στην Java, χρησιμοποιούνται σε μαθηματικές εκφράσεις, και για να περιβάλλουν τους παραμέτρους για κλήσεις μέθοδων (θα φτάσουμε σύντομα σε αυτούς).

[ ]

Αυτά είναι τετράγωνες παρενθέσεις, γνωστές στις ΗΠΑ ως απλώς brackets (βλέπετε πώς μπορεί να προκύψει σύγχυση;). Στην Java, χρησιμοποιούνται για ορισμούς πινάκων.

{ }

Αυτά είναι κυματιστές αγκύλες, επίσης γνωστή ως παραμορφωμένη παρένθεση. Στην Java, αυτά χρησιμοποιούνται για να περικυκλώσουν μπλοκ κώδικα, όπως μεθόδους ή το περιεχόμενο των κλάσεων.

Τέλος πάντων, πίσω στον κώδικα, μπορείτε να δείτε ότι δεν υπάρχει τίποτα ανάμεσα στις παραμορφωμένες

αγκύλες. Αν θέλουμε τα καβούρια μας να κάνουν το οτιδήποτε, θα πρέπει να το συμπληρώσουμε. Ας ξεκινήσουμε, με την προσθήκη μιας εντολής κίνησης στον κώδικα:

```
/**
 * Act - do whatever the Crab wants to do. This method is called whenever
 * the 'Act' or 'Run' button gets pressed in the environment.
 */
public void act()
{
    move(4);
}
```

Θα πρέπει να ταιριάζει επακριβώς ο κώδικας που είναι γραμμένος εδώ με τον δικό σας . Είναι η λέξη 'move', που ακολουθείται από στρογγυλές παρενθέσεις που περιέχει τον αριθμό 4, ακολουθούμενο από ένα ερωτηματικό.

Συνηθισμένα λάθη που μπορεί να κάνετε είναι:

- κεφαλαία γράμματα (Τα κεφαλαία έχουν μεγάλη σημασία στην Java!),
- αν λείπει το σύμβολο του ερωτηματικού,
- χρησιμοποιήσατε τις λάθος παρενθέσεις,
- κατά λάθος διαγραφή των παραμορφωμένων αγκυλών.

Εάν σας βγάζει κάποιο λάθος, αναζητήστε ένα από αυτά τα λάθη που μπορεί να έχετε κάνει κατά την αντιγραφή του κώδικα.

Μόλις γράψετε αυτόν τον κώδικα, πατήστε το κουμπί 'Μεταγλώττιση Όλων' στην κύρια διεπαφή Greenfoot (ή 'Μεταγλώττιση' στην κορυφή του επεξεργαστή). Τοποθετήστε ένα καβούρι στον κόσμο και κάντε κλικ στο κουμπί 'Εκτέλεση'. Τώρα το καβούρι πρέπει να γλιστρά στο πλάι κατά μήκος της οθόνης. Στη συνέχεια θα πρέπει να χτυπήσει την άκρη του κόσμου και να σταματήσει απότομα. Αν θέλετε, μπορείτε να διακόψετε το σενάριο, να σύρετε το καβούρι πάνω αριστερά, να πατήστε 'Εκτέλεση' και να το δείτε ξανά. Γιατί δεν προσπαθείτε επίσης να τοποθετήσετε περισσότερα από ένα καβούρια στον κόσμο και να τα παρακολουθήσετε όλα να κάνουν το ίδιο? Προσοχή: Τα καβούρια δεν σταματούν στην πραγματικότητα με αυτόν τον τρόπο. Εξακολουθούν να προσπαθούν να κινηθούν, αλλά το Greenfoot δεν τα αφήνει να κινούνται έξω από τον κόσμο (αν τα άφηνε, πώς θα τα μεταφέρετε πίσω πάλι?).

Μπορείτε να μεταβάλετε την ταχύτητα του Καβουριού, αλλάζοντας τον αριθμό 4 του κώδικα σε ένα διαφορετικό αριθμό. Με υψηλότερες τιμές θα είναι ταχύτερα, ενώ με μικρότερες θα είναι πιο αργά - δείτε αν μπορείτε να μαντέψετε τι θα συμβεί αν βάλετε έναν αρνητικό αριθμό.

Ας κάνουμε το καβούρι να κάνει λίγο περισσότερο από το να κινείται σε μια ευθεία γραμμή. Επιστρέψτε στον κώδικα, και μετά την γραμμή για την κίνηση, προσθέστε άλλη μία γραμμή (αλλά και η καινούρια γραμμή να είναι μέσα στην παραμορφωμένη παρένθεση), η οποία γραμμή θα λέει: turn(3); όπως το παρακάτω:

```
/**
 * Act - do whatever the Crab wants to do. This method is called whenever
 * the 'Act' or 'Run' button gets pressed in the environment.
 */
public void act()
{
    move(4);
    turn(3);
}
```

Θα δείτε ότι το καβούρι τρέχει σε έναν κύκλο. Πειραματιστείτε με το turn για να πάρει ο κύκλος μικρότερο η μεγαλύτερο μέγεθος - θα το αφήσουμε σε σας να καταλάβετε με ποιο τρόπο θα πρέπει να το κάνετε, και γιατί θα πρέπει να γίνει έτσι.
Το καλό πράγμα για το καβούρι που γυρίζει όλη την ώρα είναι ότι ακόμη και αν χτυπήσει την άκρη του κόσμου, τελικά θα βρεθεί στην στη μέση πάλι.

Ακόμη καλύτερο θα ήταν να μπορούμε να ελέγξουμε τη στροφή του καβουριού – Ας βάλουμε λοιπόν λίγη αλληλεπίδραση μέσα στο σενάριο μας! Μπορούμε να κάνουμε το καβούρι να στρίβει, όταν πατάμε το αριστερό ή το δεξί πλήκτρα(βέλος). Εδώ είναι ο κώδικας:

```
/**
 * Act - do whatever the Crab wants to do. This method is called whenever
 * the 'Act' or 'Run' button gets pressed in the environment.
 */
public void act()
{
    move(4);

    if (Greenfoot.isKeyDown("left"))
    {
        turn(-3);
    }
    if (Greenfoot.isKeyDown("right"))
    {
        turn(3);
    }
}
```

Χρησιμοποιούμε ενσωματωμένες μεθόδους του Greenfoot για να ελέγχουμε εάν ένα πλήκτρο είναι πατημένο. Ανάμεσα στα εισαγωγικά είναι το όνομα του πλήκτρου, "left" είναι το πλήκτρο με το αριστερό βελάκι, "right" είναι το πλήκτρο με δεξί. Εάν θέλετε κάτι σαν "a" ή "d", απλά χρησιμοποιήστε αυτά τα γράμματα αντί right και left! Ο κωδικός μας λέει: αν τα πλήκτρα είναι πατημένα, γύρισε σε ένα συγκεκριμένο αριθμό. Εισάγετε αυτόν κώδικα, μεταγλωττίστε και δοκιμάσετε το. Μπορείτε να αλλάξετε το πόσο γρήγορα ο κάβουρας γυρίζει με την αύξηση αυτών των αριθμών.

Αν βάλετε πολλά καβούρια στον κόσμο, θα δείτε ότι πατώντας τα πλήκτρα ελέγχου όλα μαζί ταυτόχρονα με σωστό συγχρονισμό, θα υπάρξει υπερφόρτωση: όλα τα καβούρια εκτελούν τον ίδιο κώδικα, έτσι ώστε αν πατήσετε το αριστερό πλήκτρο, όλα τους θα δούν ότι το αριστερό πλήκτρο είναι πατημένο, και όλοι θα μετατραπούν αναλόγως.

Τι θα συμβεί αν κρατάτε πατημένο το αριστερό και το δεξί την ίδια στιγμή; Δοκιμάστε το και ανακαλύψτε - στη συνέχεια εξετάστε τον κώδικα και δείτε αν μπορείτε να ανακαλύψετε το λόγο που συμβαίνει αυτό.

Ας συνεχίσουμε με το μέρος 3 του tutorial.

ΜΕΡΟΣ 3

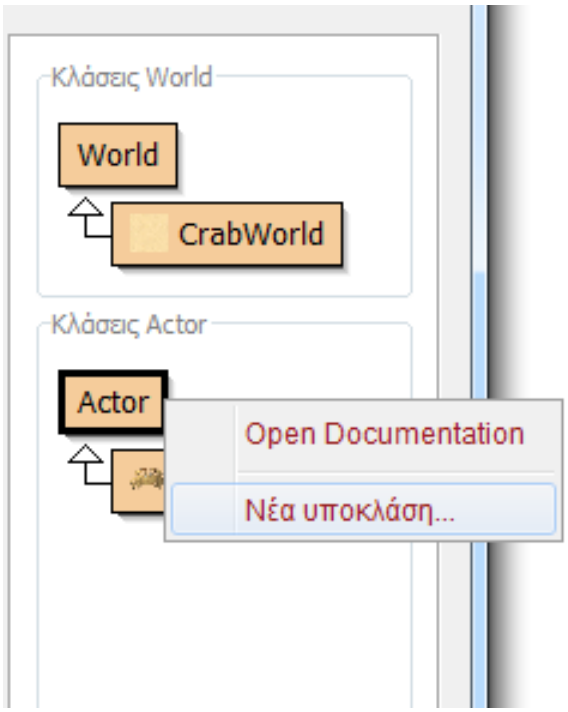
Tutorial 3: Εντοπισμός και Αφαίρεση των Actors, και επίσης πως γράφουμε μεθόδους

Σε αυτό το tutorial θα εξηγήσουμε πώς να μάθετε εάν έχετε επαφή με έναν άλλο Actor, και στη συνέχεια να αφαιρέσετε αυτόν τον Actor από τον κόσμο. Επίσης, θα εξηγήσουμε πώς να κρατήσετε τον κώδικα σας ευανάγνωστο χρησιμοποιώντας μεθόδους.

Τρώγοντας Σκουλήκια

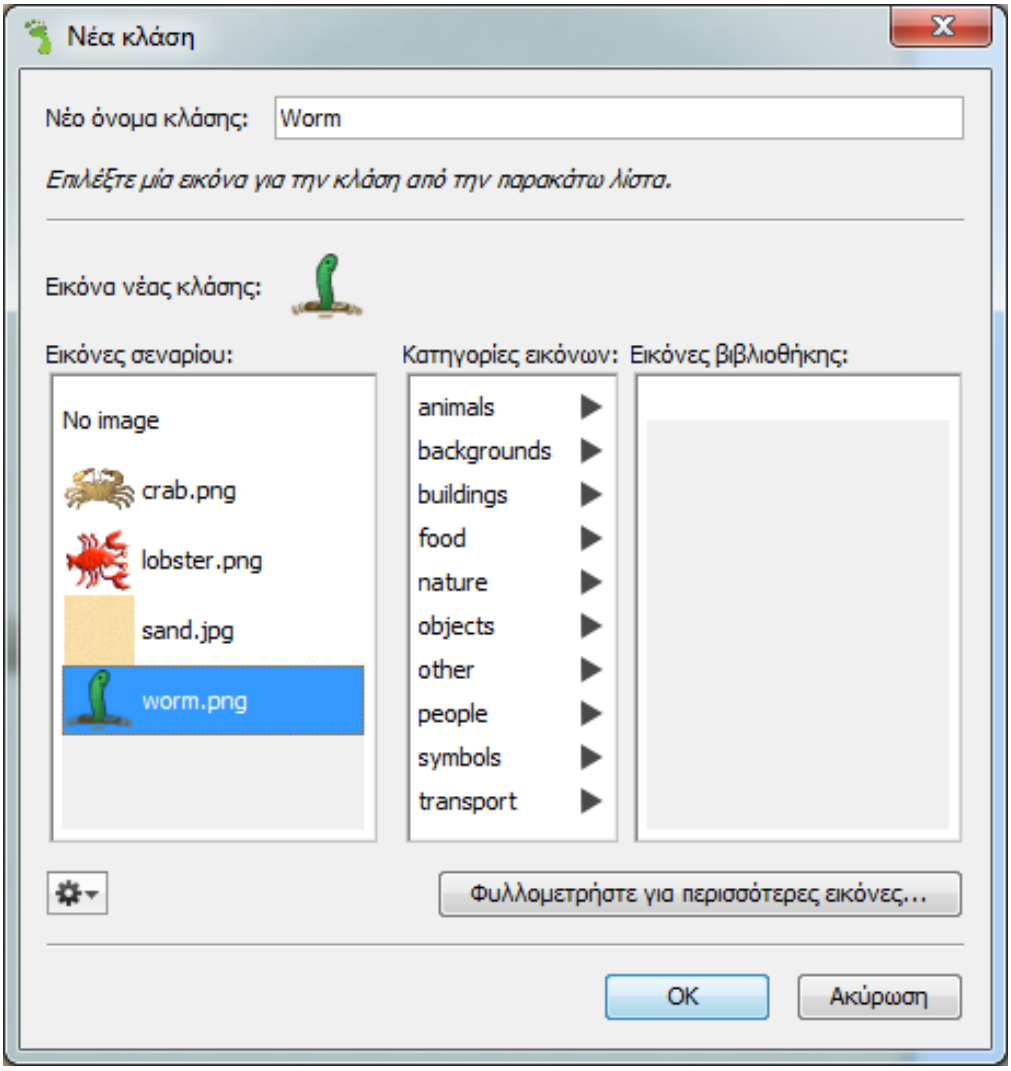
Θα συνεχίσουμε από το μέρος 2, στο οποίο βάλαμε τα καβούρια μας να κινούνται γύρω από την οθόνη, υπό τον έλεγχό μας. Σε αυτό το μέρος, θα πάμε να εισάγουμε μερικά σκουλήκια για να φάμε.

Έχουμε ήδη μια κλάση για το καβούρι - είμαστε τώρα έτοιμοι να προσθέσουμε άλλη μία κλάση για το σκουλήκι. Το Σκουλήκι πρόκειται επίσης να είναι ένας Actor, έτσι λοιπόν για να φτιάξετε την κλάση σκουλήκι (Worm στα Αγγλικά), κάντε δεξί κλικ στην κατηγορία Actor και επιλέξτε 'νέα υποκλάση ...':



Ως νέο όνομα της κλάσης, πληκτρολογήστε: Worm - σημειώστε ότι γράφουμε το W με κεφαλαίο. Είναι συνήθεια στην Java να αρχίζουμε τα ονόματα κλάσεων με κεφαλαίο γράμμα. Εάν εισάγετε κατά λάθος ένα μικρό 'w' τώρα, θα πάρετε ένα λάθος αργότερα κατά την αντιγραφή του κώδικα μας που έχει χρησιμοποιηθεί κεφάλαιο 'W'.

Από την αριστερή λίστα των εικόνων, επιλέξτε την "worm.png" ως εικόνα και πατήστε Ok:



Η κλάση 'Worm' δεν θα έχει πραγματικό κώδικα για την ώρα- ακριβώς όπως ήταν η κλάση Καβούρι όταν ξεκινήσαμε. Θα το αφήσουμε έτσι. Τα Σκουλήκια μας είναι χαζά και απλά κάθονται ακίνητα σε ένα μέρος, έτοιμα να φαγωθούν. Εύκολη λεία! Εμείς πάμε για να τροποποιήσουμε την κλάση καβούρι, έτσι ώστε όταν τα καβούρια μας περάσουν πάνω από ένα σκουλήκι, να το τρώνε. Για να γίνει αυτό, πάμε πίσω στον πηγαίο κώδικα του καβουριού μας, που τώρα μοιάζει με αυτό:

```
/**
 * Act - do whatever the Crab wants to do. This method is called whenever
 * the 'Act' or 'Run' button gets pressed in the environment.
 */
public void act()
{
    move(4);

    if (Greenfoot.isKeyDown("left"))
    {
        turn(-3);
    }
    if (Greenfoot.isKeyDown("right"))
    {
        turn(3);
    }
}
```

Στο τέλος της μεθόδου ‘act()’, θα εισάγετε κώδικα για να ελέγξετε αν το καβούρι αγγίζει ένα σκουλήκι. Θα χρησιμοποιήσουμε την μέθοδο getOneObjectAtOffset(). Αυτή η μέθοδος παίρνει τρεις παραμέτρους. Οι δύο πρώτες είναι το X και Y offset (διαφορά) από την τρέχουσα θέση μας. Θέλουμε να γνωρίζουμε τι ακριβώς είναι από κάτω μας, έτσι θα περάσουμε το μηδέν και για τις δύο από αυτές τις παραμέτρους. Η τρίτη παράμετρος, μας επιτρέπει να αναφέρουμε ποια κλάση μας ενδιαφέρει. Η κλάση αυτή είναι η κλάση Worm:

```
/**
 * Act - do whatever the Crab wants to do. This method is called whenever
 * the 'Act' or 'Run' button gets pressed in the environment.
 */
public void act()
{
    move(4);

    if (Greenfoot.isKeyDown("left"))
    {
        turn(-3);
    }
    if (Greenfoot.isKeyDown("right"))
    {
        turn(3);
    }

    Actor worm;
    worm = getOneObjectAtOffset(0, 0, Worm.class);

}
```

Σημειώστε ότι καλούμε τη μέθοδο, αλλά κάνουμε και κάτι και στην αριστερή πλευρά επίσης της εντολής μας. Αυτή η μέθοδος επιστρέφει κάτι (το αντικείμενο που είναι από κάτω από μας, εφόσον υπάρχει), έτσι πρέπει να αποθηκεύσουμε αυτό το αντικείμενο που επιστρέφει και να είμαστε έτοιμοι να το χρησιμοποιήσουμε ξανά.

Για να το κάνουμε αυτό δηλώνουμε μια μεταβλητή, που την έχουμε ονομάσει *worm*, μια γραμμή πιο πριν. Στη συνέχεια, χρησιμοποιούμε τον τελεστή εκχώρησης (το σύμβολο του ίσον ‘’=’’) για να δείξουμε ότι η τιμή του σκουληκιού θα πρέπει να είναι ίση με την τιμή επιστροφής της μεθόδου.

Τώρα πια, έχουμε την τιμή επιστροφής της μεθόδου στη μεταβλητή *worm*. Αν δεν υπήρχε Σκουλήκι το οποίο μας αγγίζει, αυτή η μεταβλητή θα περιέχει την ειδική τιμή *null*. Η μόνη περίπτωση που μπορούμε να καταργήσουμε το σκουλήκι, είναι όταν υπάρχει ένα σκουλήκι, γι 'αυτό χρειαζόμαστε έναν έλεγχο ότι η μεταβλητή worm δεν είναι ίση με την ειδική τιμή *null* πριν το αφαιρέσουμε:

```
/**
 * Act - do whatever the Crab wants to do. This method is called whenever
 * the 'Act' or 'Run' button gets pressed in the environment.
 */
public void act()
{
    move(4);

    if (Greenfoot.isKeyDown("left"))
    {
        turn(-3);
    }
    if (Greenfoot.isKeyDown("right"))
    {
        turn(3);
    }

    Actor worm;
    worm = getOneObjectAtOffset(0, 0, Worm.class);
    if (worm != null)
    {
        World world;
        world = getWorld();
        world.removeObject(worm);
    }
}
```

Το σύμβολο “! =” είναι το "δεν είναι ίσο με" τελεστής στην Java. Αυτό λοιπόν σημαίνει ότι ο κώδικας θα εκτελέσει μόνο όταν το σκουλήκι δεν έχει τιμή *null.* Ο κώδικας που εισάγαμε μέσα στον έλεγχο δουλεύει ως εξής: Πρώτα ορίζουμε μια μεταβλητη (με όνομα world) ώστε να μπορέσουμε να ζητήσουμε από τον κόσμο μας στην συνέχεια να αφαιρέσει ένα αντικείμενο (δηλαδή το worm). Έπειτα αρχικοποιούμε αυτήν την μεταβλητή με τον συγκεκριμένο κόσμο που έχουμε. Αυτό το κάνει η εντολή ‘getWorld()’. Ακολούθως αφαιρούμε το σκουλήκι μας από τον κόσμο με την μέθοδο ‘removeObject()’

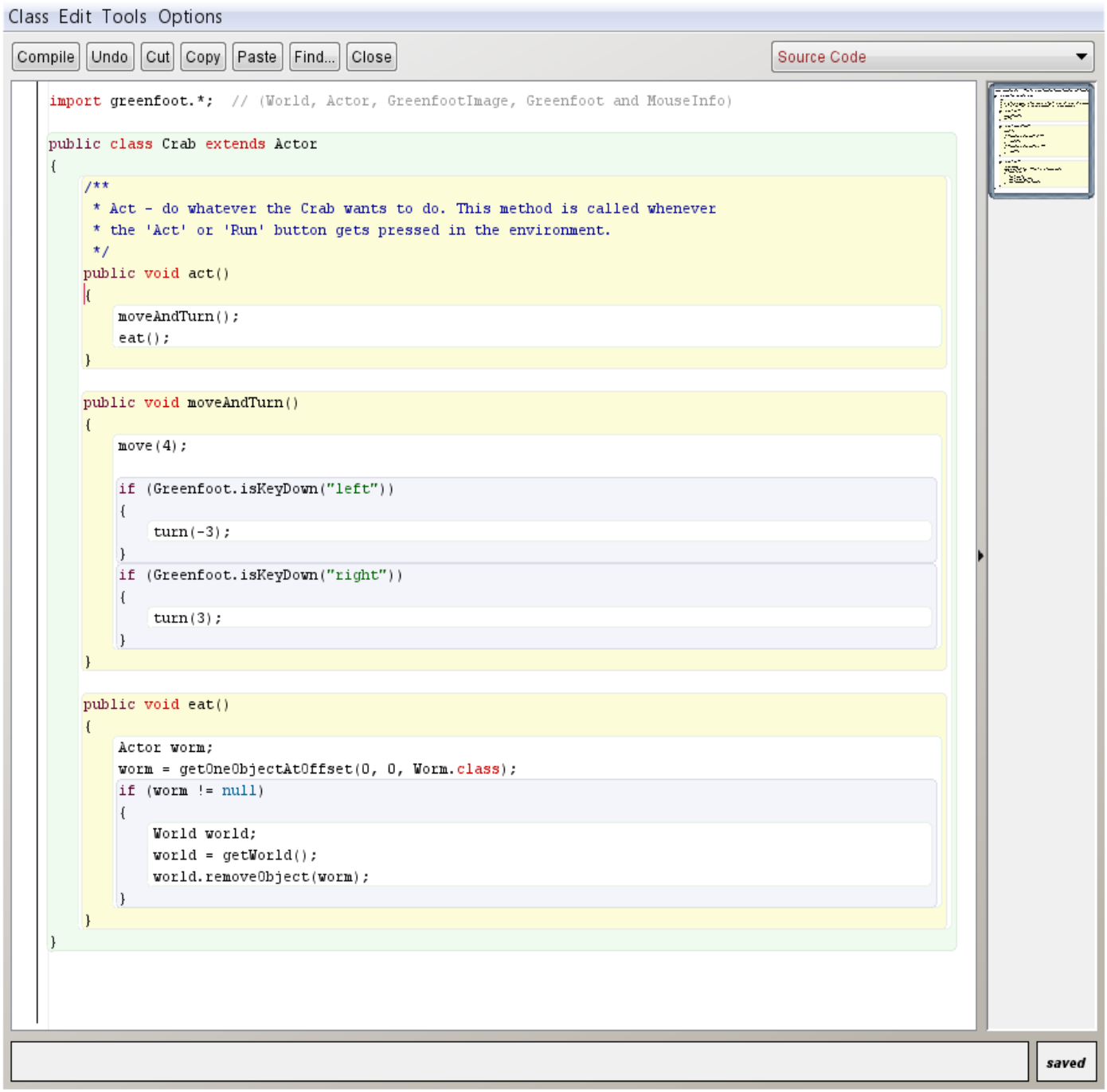
Λοιπόν, ας κάνουμε μια δοκιμή! Κάντε ‘Μεταγλώττιση’, δημιουργήστε κάποια σκουλήκια και βάλτε τα στον κόσμο σας, στη συνέχεια, προσθέστε ένα καβούρι, πατήστε ‘Εκτέλεση’ και χρησιμοποιήστε τα αριστερά και δεξιά βελάκια σας για να καθοδηγήσετε το καβούρι προς τα σκουλήκια, τα οποία θα πρέπει να φαγωθούν. Νόστιμο ε;!

Refactoring – ‘Η αλλιώς ανακατασκευή του κώδικα μας

Πριν προχωρήσουμε, θα πρέπει να κάνουμε μια αλλαγή στον κώδικα. Αυτό είναι κάτι που είναι γνωστό ως refactoring: δηλαδή κάνουμε αλλαγή του κώδικα χωρίς να αλλάξουμε την συμπεριφοράς του. Επί του παρόντος η μέθοδος ‘act()’ για το Καβούρι έχει δύο διαφορετικές συμπεριφορές μέσα της: οι μισές συμπεριφορές έχουν να κάνουν με την πέρα δώθε κίνηση, και οι άλλες μισές έχουν να κάνουν με το φάγωμα των σκουληκιών. Θα ήταν σαφέστερο να τις ονομάσουμε και θα τις χωρίσουμε, για να αποφύγουμε να συγχύσουμε τους εαυτούς μας αργότερα. Αυτό μπορούμε να το καταφέρουμε με τη δημιουργία διαφορετικών μεθόδων για κάθε συμπεριφορά. Στην πραγματικότητα έχουμε ήδη δει πώς γράφονται οι μέθοδοι: στο κάτω, κάτω η act() είναι και αυτή μια μέθοδος.

Και οι δύο νέες μέθοδοι μας δεν απαιτούν παραμέτρους και δεν επιστρέφουν καμία τιμή, άρα μπορούν να δηλώνονται ακριβώς με τον ίδιο τρόπο όπως και η ‘act()’;

Ο αλλαγμένος κώδικας μας φαίνεται παρακάτω:



Αν συγκρίνετε αυτόν τον κώδικα με την προηγούμενη του έκδοση, θα παρατηρήσετε ότι δεν έχουμε αλλάξει ή αφαιρέσει οποιοδήποτε εντολή. Αυτό που κάναμε ήταν να μετακινήσουμε το πάνω μισό σε μια νέα μέθοδο που ονομάζεται ‘moveAndTurn’ (δηλαδή, κινήσου και στρίψε), και μετά μετακινήσαμε το κάτω μισό σε μια νέα μέθοδο που ονομάζεται ‘eat’ (δηλαδή, φάε). Μετά αντικαταστήσαμε το περιεχόμενο της μεθόδου ‘act()’ με τις κλήσεις σε αυτές τις νέες μεθόδους. Αυτό θα κάνει τον κώδικα που έχουμε, να επικεντρωθεί σε μικρότερα τμήματα στην συνέχεια του tutorial. Το μέρος 4 ακολουθεί.

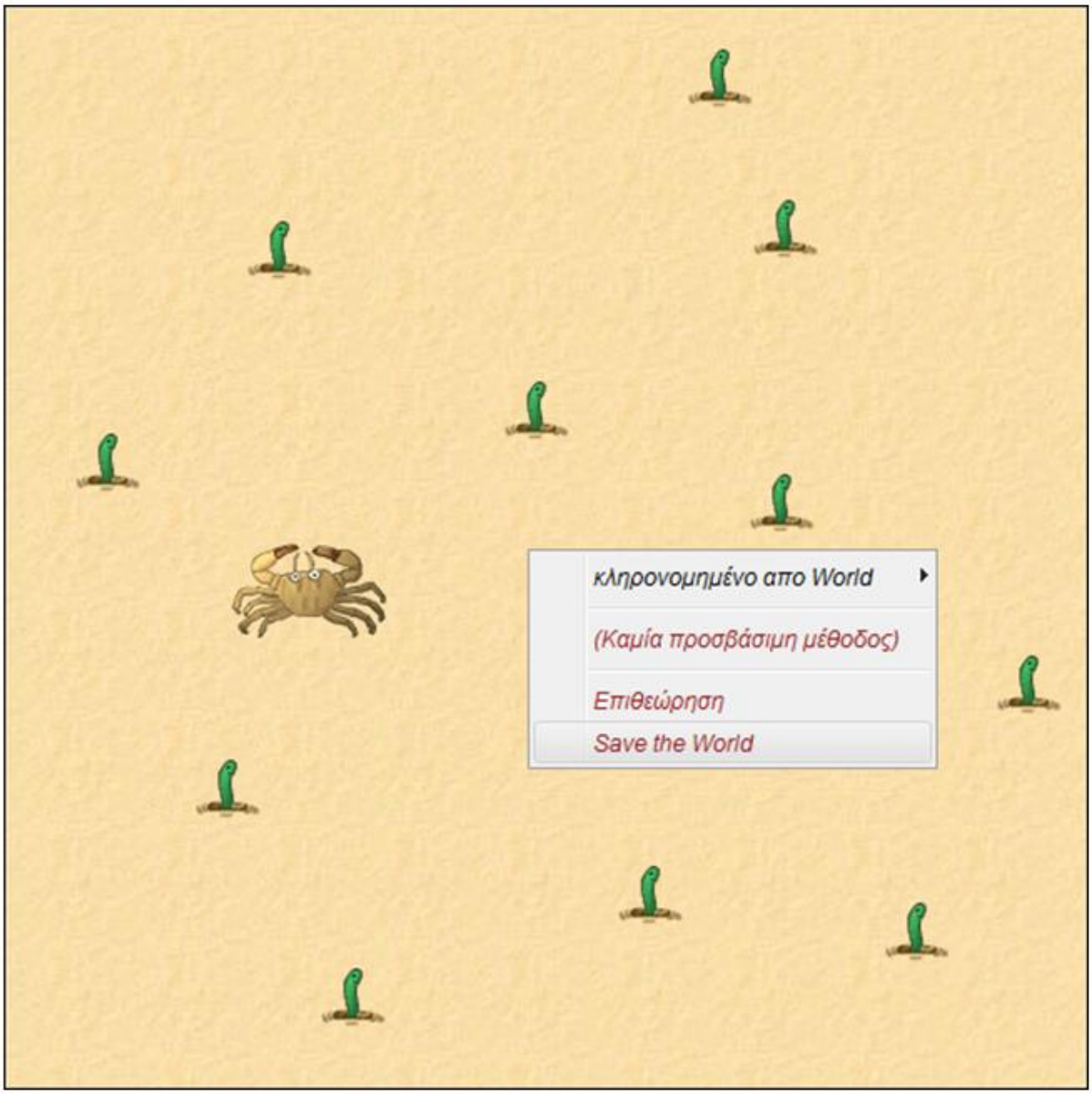
ΜΕΡΟΣ 4

Tutorial 4: Αποθηκεύοντας τον Κόσμο, Φτιάχνοντας και αναπαράγοντας ήχο

Αυτό το tutorial θα εξηγήσει πώς να αρχικοποιήσετε στον κόσμο τους Actors, καθώς και πώς να παίζετε και να καταγράφετε ήχους.

"Αποθηκεύοντας" τον κόσμο

Μέχρι τώρα έχετε σίγουρα κουραστεί να χρειάζεται να προσθέσετε νέα αντικείμενα στον κόσμο κάθε φορά που πρέπει να μεταγλωττίσουμε τον κώδικα. Είναι εφικτό να προσθέσουμε λίγο κώδικα για να δημιουργήσετε αυτόματα κάποια από τα σκουλήκια και τα καβούρια. Επιπλέον, είναι δυνατόν να κάνετε το Greenfoot να γράψει αυτόν τον κώδικα για εσάς! Πατήστε Επαναφορά(reset), στη συνέχεια, προσθέστε σκουλήκια και ένα καβούρι στον κόσμο. Πριν πατήσετε 'Εκτέλεση', κάντε δεξί κλικ πάνω στον κόσμο και να επιλέξετε την επιλογή: 'Save the World' (δηλαδή, αποθηκεύσετε τον κόσμο):



Αυτό προσθέτει λίγο κώδικα στην κλάση CrabWorld, η οποία θα δημιουργήσει τα σκουλήκια και το καβούρι και θα τα τοποθετήσει στον κόσμο στην ίδια θέση την επόμενη φορά που θα πατήσετε επαναφορά ή θα μεταγλωττίσετε. Δεν θα εξερευνήσουμε τον κώδικα λεπτομερώς αυτή την στιγμή, αλλά αν είστε περίεργοι τότε μπορείτε να κοιτάξετε στον πηγαίο κώδικα της κλάσης CrabWorld.

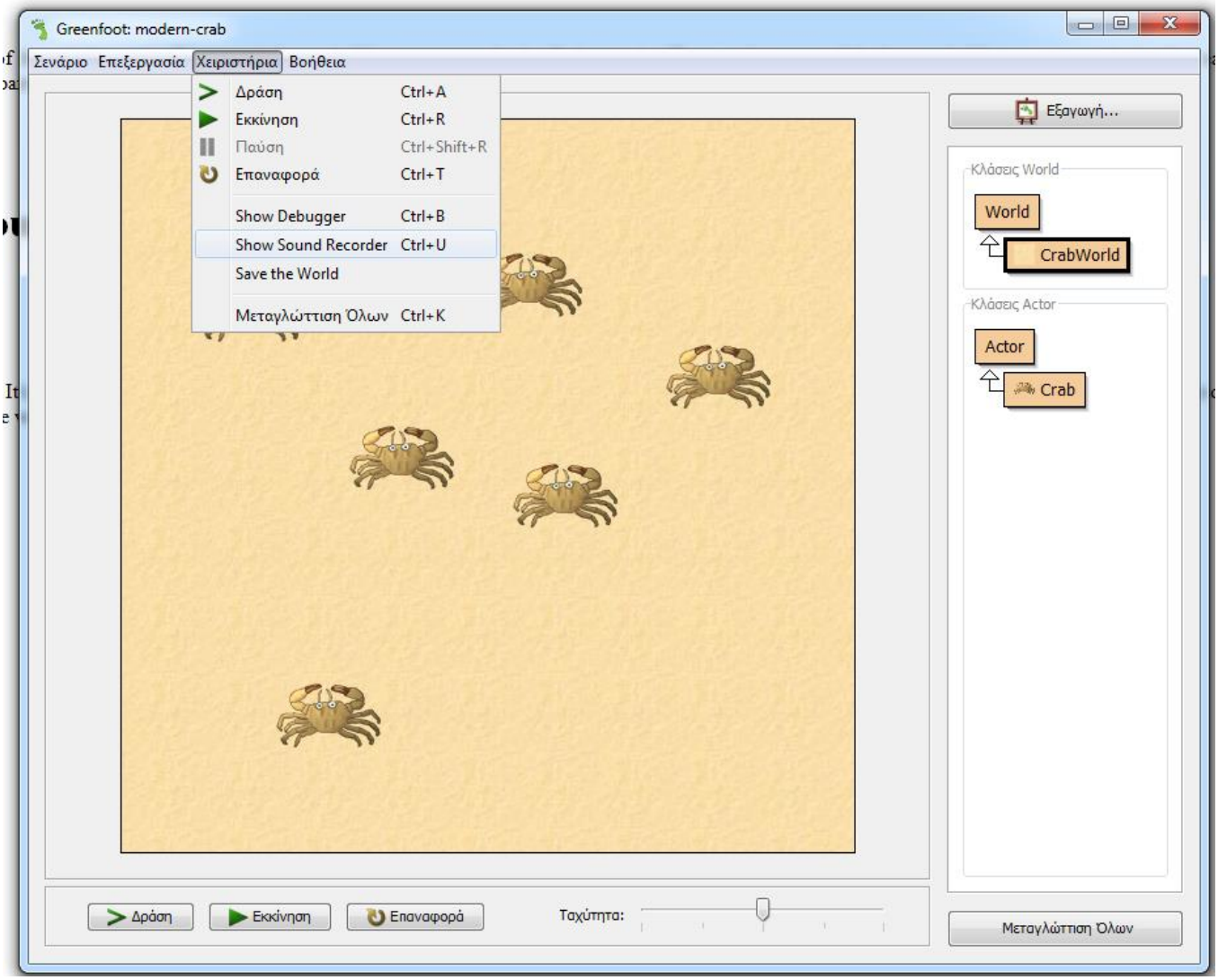
Αναπαραγωγή και εγγραφή ήχου

Μπορούμε να προσθέσουμε κάποιον ήχο στο σενάριο μας. Το σενάριο έχει έναν ήχο έτοιμο για να τον χρησιμοποιήσετε, με το όνομα "eating.wav". Μπορούμε να κάνουμε αυτόν τον ήχο να παίζει κάθε φορά που το καβούρι τρώει ένα σκουλήκι, με την προσθήκη μιας μονάχα γραμμής στην κλάση καβούρι. Η εντολή αυτή είναι η 'Greenfoot.playSound()'.

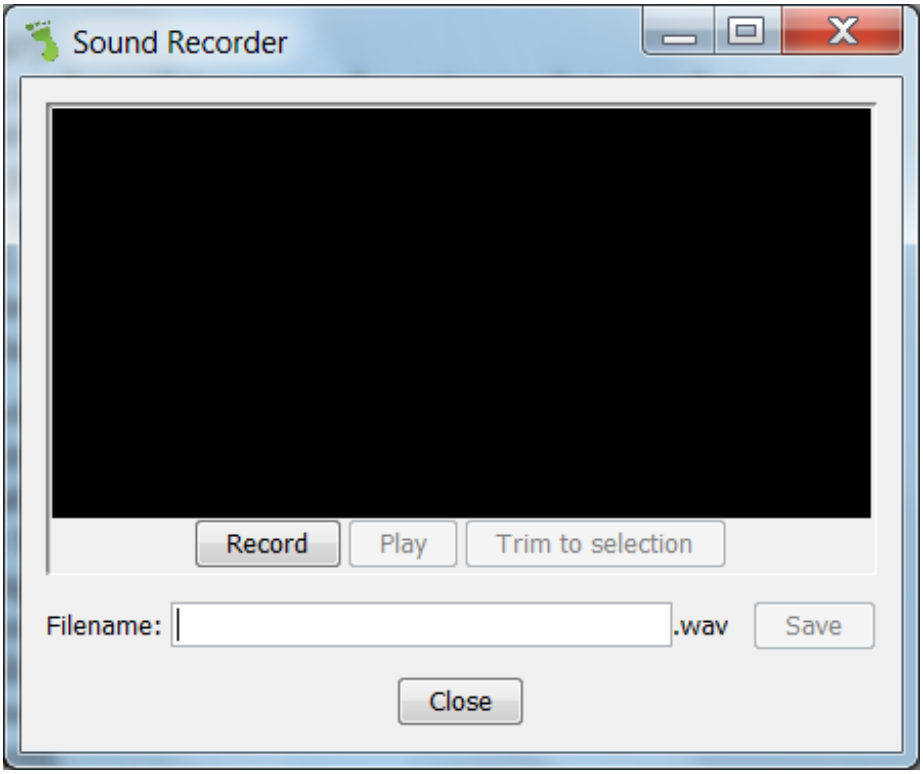
```
public void eat()
{
    Actor worm;
    worm = getOneObjectAtOffset(0, 0, Worm.class);
    if (worm != null)
    {
        World world;
        world = getWorld();
        world.removeObject(worm);
        Greenfoot.playSound("eating.wav");
    }
}
```

Δοκιμάστε το! Βέβαια μην ξεχάσετε να ανοίξετε τα ηχεία σας! (ή να συνδέσετε τα ακουστικά σας).

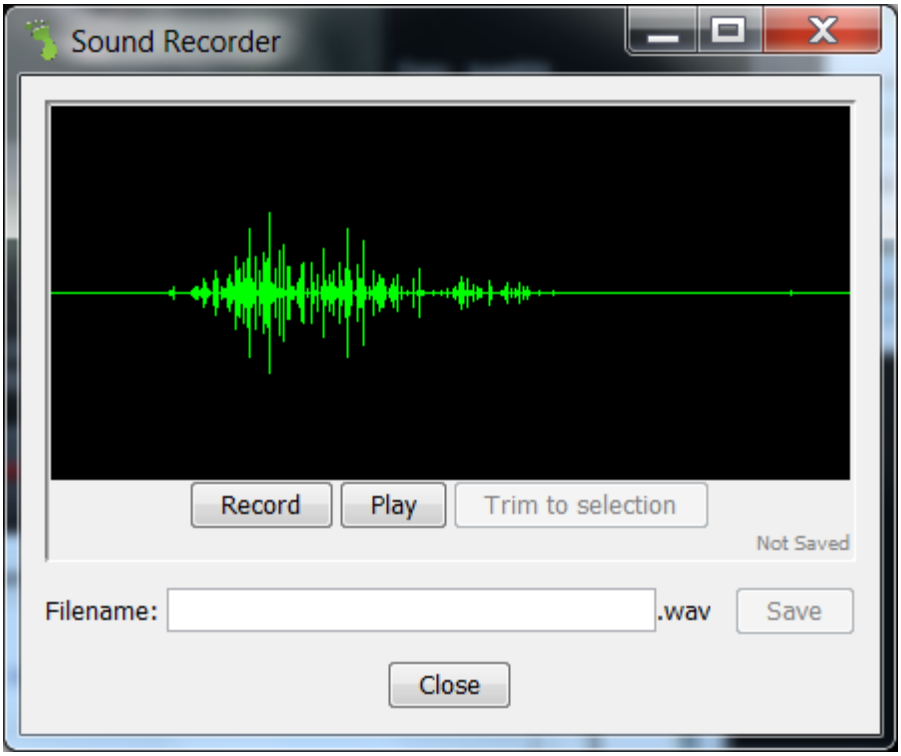
Κάτι τελευταίο – Εάν διαθέτεται ένα μικρόφωνο στον υπολογιστή σας, μπορείτε κάλλιστα να ηχογραφήσετε τους δικούς σας ήχους. Στο μενού επιλογών υπάρχει μια επιλογή που επιτρέπει την εμφάνιση της λειτουργίας ηχογράφησης ήχου. Επιλέξτε την κάνοντας κλικ στο 'Show Sound Recorder':



Τώρα μπορείτε να κάνετε εγγραφή ήχου:



Πατήστε το κουμπί εγγραφής και μιλήστε. Στη συνέχεια, πατήστε το κουμπί για να σταματήσει. Θα πρέπει να βλέπετε ένα πράσινο κύμα, και όταν πατάτε αναπαραγωγή θα πρέπει να ακούσετε το θόρυβο που μόλις ηχογραφήσατε. Αν όχι, τότε ίσως υπάρχει πρόβλημα με το μικρόφωνό σας. Υποθέτοντας ότι παίζει τον ήχο, θα έχετε σχεδόν πάντα ένα κομμάτι σιωπής στην αρχή και στο τέλος του ήχου - μπορείτε να το δείτε αυτό στην πράσινη οθόνη, καθώς θα υπάρχει μια επίπεδη οριζόντια γραμμή στην αρχή και στο τέλος του σχήματος:



Η σιωπή στο τέλος δεν είναι μεγάλο πρόβλημα, αλλά η σιωπή στην αρχή είναι σίγουρα ενοχλητική - αυτό σημαίνει ότι όταν λέτε στον ήχο να αναπαράγεται κάθε φορά που ένα καβούρι τρώει ένα σκουλήκι, θα υπάρχει μια μικρή καθυστέρηση πριν ο ήχος αρχίζει να παίζει. Μπορείτε να αφαιρέσετε τη σιωπή επιλέγοντας το κομμάτι στη μέση (το κομμάτι που θέλετε να κρατήσετε), κάνοντας κλικ στην αρχή (μετά την αρχική σιωπή) και σύροντας προς το τέλος (πριν από την τελική σιωπή) - η επιλογή θα πρέπει να φαίνεται σε γκρι χρώμα. Στη συνέχεια πατήστε 'Trim to selection'. Η σιωπή θα αφαιρεθεί.

Αποθηκεύστε τον ήχο εισάγοντας την ονομασία του αρχείου (π.χ. "myeating") και στην συνέχεια πατώντας το πλήκτρο 'Save' (δηλαδή, αποθήκευση). Κλείστε την ηχογράφηση και επιστρέψτε στον κώδικά σας. Βρείτε τη γραμμή με το "eating.wav" και αλλάξτε την σε "myeating.wav" (ή οποιοδήποτε όνομα χρησιμοποιήσατε, χωρίς να ξεχάσετε να βάλετε και την επέκταση .wav). Στη συνέχεια, όταν παίζετε το παιχνίδι σας, θα πρέπει να ακούσετε το δικό σας ήχο.

Είμαστε κοντά στην ολοκλήρωση του παιχνιδιού μας, αλλά πρέπει πρώτα να προσθέσουμε και έναν εχθρό, κάτι το οποίο θα κάνουμε στο μέρος 5.

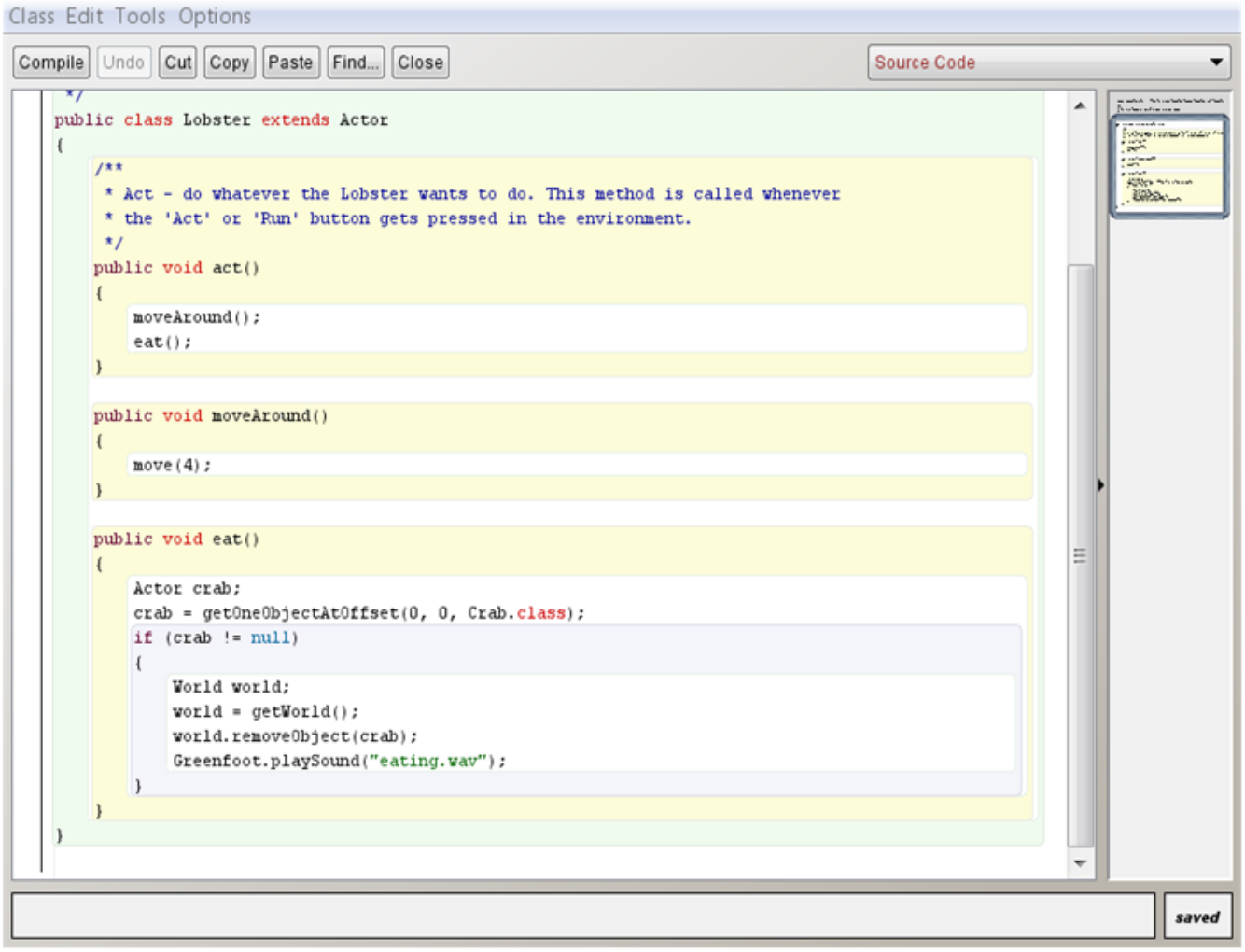
ΜΕΡΟΣ 5

Tutorial 5: Προσθέτοντας έναν τυχαίο κινούμενο εχθρό

Μετά και από το τελευταίο tutorial μας, έχουμε πια ένα σενάριο με ένα καβούρι που μπορούμε να ελέγξουμε, που τρέχει γύρω-γύρω και τρώει σκουλήκια. Το παιχνίδι είναι αρκετά εύκολο βέβαια - ακόμα κι αν ο κάβουρας (εσκεμμένα) είναι λίγο δύσκολο να ελεγχτεί – δεν υπάρχει αρκετή ένταση. Αυτό που χρειαζόμαστε είναι ένας εχθρός που τρώει τα καβούρια: ένας αστακός!

Κατ 'αρχήν, ας προσθέσουμε ένα αστακό που κινείται σε ευθεία γραμμή και τρώει καβούρια. Μπορούμε να το κάνουμε αυτό χρησιμοποιώντας τον κώδικα που γνωρίζουμε ήδη.

Πρώτον, προσθέτουμε μια κλάση που την ονομάζουμε Αστακός - θυμηθείτε ότι το κάνουμε αυτό με δεξί κλικ στην κλάση Actor και επιλέγοντας νέα υποκλάση. Θα βρείτε την εικόνα του αστακού στη λίστα των εικόνων στα αριστερά. Αφού έχετε κάνει την κλάση Αστακός, μπορείτε να την κάνετε να κινείται σε ευθεία γραμμή και να τρώει ένα καβούρι, εάν πετύχει στο δρόμο του ένα. Έχετε ήδη δει πώς να τα κάνετε όλα αυτά στα προηγούμενα tutorials, γι 'αυτό δεν θα επικεντρωθούμε σε αυτά τώρα. Αν τώρα έχετε κολλήσει σε αυτό, μπορείτε να δείτε την απάντηση στην εικόνα παρακάτω. Όμως σας προτείνουμε να δοκιμάσετε μόνοι σας.




```
*/
public class Lobster extends Actor
{
    /**
     * Act - do whatever the Lobster wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        moveAround();
        eat();
    }

    public void moveAround()
    {
        move(4);
    }

    public void eat()
    {
        Actor crab;
        crab = getOneObjectAtOffset(0, 0, Crab.class);
        if (crab != null)
        {
            World world;
            world = getWorld();
            world.removeObject(crab);
            Greenfoot.playSound("eating.wav");
        }
    }
}
```

Μπορείτε να ελέγξετε ότι ο αστακός λειτουργεί, με το να τοποθετήσετε ένα αστακό στα αριστερά του κάβουρα, στη συνέχεια, επιλέξτε 'Εκτέλεση' και αφήστε τους να τρέχουν προς τη δεξιά πλευρά του κόσμου, όπου ο κάβουρας θα φαγωθεί. (Αν το θέλετε, φτιάξτε το δικό σας ήχο για το όταν ο αστακός τρώει τα καβούρια.)

Ωραία τώρα το σενάριο μας είναι καλό όμως ο αστακός μας είναι αρκετά χαζός. Είναι πολύ εύκολο να ξεφύγουμε από αυτόν με την κίνηση προς τα πλάγια. Επίσης, όταν φτάνει τη δεξιά άκρη του κόσμου, μένει εκεί για πάντα (ακριβώς όπως τα καβούρια μας έκαναν στην αρχή).

Ας κάνουμε τον αστακό μας πιο δύσκολο να τον αποφύγουμε με την εισαγωγή κάποιας τυχαίας κίνησης. Το Greenfoot παρέχει μια λειτουργία, την: 'Greenfoot.getRandomNumber' που θα σας δώσει έναν τυχαίο αριθμό. Εδώ είναι μια πρώτη προσπάθεια, όπου θα μπορεί να αλλάζει με μια τυχαία τιμή σε κάθε καρέ:

```
public void moveAround()
{
    move(4);
    turn(Greenfoot.getRandomNumber(90));
}
```

Ο κώδικας αυτός σημαίνει ότι θα γυρίζει σε μία τυχαία τιμή σε κάθε καρέ, μεταξύ 0 και 90 μοίρες. Δοκιμάστε το, και δείτε πώς λειτουργεί.

Θα δείτε ότι αυτό δεν δημιουργεί ένα πολύ απειλητικό εχθρό: ο αστακός φαίνεται να περιστρέφεται ως επί το πλείστον σε ένα σημείο και γυρίζει πάντα προς τα δεξιά. Ας διορθώσουμε αυτά τα προβλήματα ένα-ένα, ξεκινώντας με την περιστροφή σε ένα σημείο.

Αυτή τη στιγμή, ο αστακό περιστρέφεται σε κύκλους άρα περιπλανιέται τυχαία. Υπάρχουν μερικοί τρόποι για να τον κάνουμε να γυρίζει σε κύκλους λιγότερο συχνά. Ένας θα ήταν να έχει έναν μετρητή που καταγράφει το χρονικό διάστημα που πέρασε από την τελευταία στροφή, ας πούμε, κάθε 10 καρέ. Ένας άλλος τρόπος είναι να χρησιμοποιήσουμε μια γεννήτρια τυχαίων αριθμών για να γυρίζει τυχαία, με μια ορισμένη συχνότητα (π.χ. κάθε 10 καρέ). Ας δουμε την χρήση της γεννήτριας τυχαίων αριθμών, για να κάνουμε ένα λιγότερο προβλέψιμο αστακό.

Ας πούμε ότι ένας αστακός έχει 10% πιθανότητα να κάνει στροφή σε κάθε καρέ. Μπορούμε να το υλοποιήσουμε αυτό χρησιμοποιώντας την εντολή 'Greenfoot.getRandomNumber(100)' που επιστρέφει έναν τυχαίο αριθμό από το 1 έως το 100. Η πιθανότητα, αυτή η εντολή να επιστρέψει έναν αριθμό μικρότερο του 10 είναι φυσικά 10%. Δείτε παρακάτω:

```
public void moveAround()
{
    move(4);
    if (Greenfoot.getRandomNumber(100) < 10)
    {
        turn(Greenfoot.getRandomNumber(90));
    }
}
```

Σκεφτείτε προσεκτικά το πώς λειτουργεί αυτό και γιατί χρησιμοποιούμε < αντί για <=. Θα μπορούσαμε να το υλοποιήσουμε αυτό με διαφορετικό τρόπο; Για παράδειγμα χρησιμοποιώντας

Greenfoot.getRandomNumber(50) ή Greenfoot.getRandomNumber(10); Τι γίνεται με: Greenfoot.getRandomNumber(5);

Αυτό λοιπόν θα κάνει τον αστακό μας να γυρίζει (κατά μέσο όρο) κάθε 10 καρέ. Μεταγλωττίστε και εκτελέστε το, και δείτε τι συμβαίνει. Ο αστακός πρέπει κυρίως να περιπλανιέται κατά μήκος σε μια ευθεία γραμμή, περιστασιακά στρίβοντας δεξιά. Αυτό είναι καλό, και μας φέρνει πίσω στο άλλο πρόβλημά μας: ο αστακός γυρίζει πάντα δεξιά.

Γνωρίζουμε από το καβούρι μας ότι ο τρόπος για να στρίψει αριστερά είναι να χρησιμοποιήσουμε έναν αρνητικό αριθμό για τη γωνία. Αν μπορούσαμε να αλλάξουμε τη στροφή του αστακού μας από την εμβέλεια 0 έως 90 στη εμβέλεια -45 έως 45, θα έφτιαχνε το πρόβλημα μας.

Υπάρχουν αρκετοί διαφορετικοί τρόποι για να επιτευχθεί αυτό, αλλά αυτός είναι ο απλούστερος: Κοιτάξτε ότι εάν αφαιρέσουμε από τον αριθμό το 45, θα καταλήξουμε με έναν αριθμό στη σωστή εμβέλεια. Ας προσαρμόσουμε λοιπόν ανάλογα τον κώδικα μας:

```
public void moveAround()
{
    move(4);
    if (Greenfoot.getRandomNumber(100) < 10)
    {
        turn(Greenfoot.getRandomNumber(90) - 45);
    }
}
```

Είναι η εμβέλεια μας για να γυρίζει ο αστακός απόλυτα συμμετρική αυτή τη στιγμή; Αν όχι, πώς θα μπορούσατε να το διορθώσουμε αυτό;

Μεταγλωττίστε και εκτελέστε το, και τώρα θα πρέπει να έχουμε ένα αποτελεσματικό αρπακτικό ζώο που μπορεί να γυρίσει προς το μέρος μας ανά πάσα στιγμή. Βάλτε ένα καβούρι, μερικούς αστακούς και πολλά, πολλά σκουλήκια στον κόσμο σας. Μη ξεχνάτε ότι μπορείτε να αποθηκεύστε τον κόσμο αν θέλετε.

Προσπαθήστε να φάτε όλα τα σκουλήκια, πριν σας πιάσει ένας αστακός. Μπορείτε να παρατηρήσετε, ωστόσο, ότι υπάρχει ένα ακόμα ελάττωμα: οι αστακοί μπορεί να κολλήσουν για ένα διάστημα στις άκρες του κόσμου. Αυτό οφείλεται στο γεγονός ότι από τη στιγμή που χτύπησε την άκρη του κόσμου, θα κινηθεί μακριά από εκεί μόνο μετά από μερικές τυχαίες στροφές.

Μπορούμε να επιταχύνουμε τη διαδικασία του να φεύγουν οι αστακοί από τις άκρες, κάνοντας τους στροφή 180 μοίρες μόλις φτάσουν στην άκρη του κόσμου. Μπορούμε να ελέγξουμε αν είναι στην άκρη του κόσμου με το να κοιτάξουμε αν η X συντεταγμένη τους είναι κοντά στο μηδέν, ή κοντά στο πλάτος του κόσμου - και παρόμοια λογική για την Y συντεταγμένη (και το ύψος του κόσμου). Ο κώδικας είναι παρακάτω:

```
public void moveAround()
{
    move(4);
    if (Greenfoot.getRandomNumber(100) < 10)
    {
        turn(Greenfoot.getRandomNumber(90) - 45);
    }
    if (getX() <= 5 || getX() >= getWorld().getWidth() - 5)
    {
        turn(180);
    }
    if (getY() <= 5 || getY() >= getWorld().getHeight() - 5)
    {
        turn(180);
    }
}
```

Και εδώ λοιπόν ολοκληρώσαμε το σενάριο με τα καβούρια. Φυσικά μπορείτε να συνεχίσετε να πειραματίζεστε με αυτό και να προσθέτετε νέα ενδιαφέροντα χαρακτηριστικά (όπως είναι ένας δεύτερος παίκτης), ή να χρησιμοποιήσετε τις γνώσεις σας για να φτιάξετε το δικό σας νέο σενάριο.

Καλή σας διασκέδαση!