

# Τα πρώτα βήματα με το Greenfoot

Προγραμματίστε το πρώτο σας  
παιχνίδι σε Java



Νικολός Δημήτρης

## 1. Πρόλογος

Το Greenfoot είναι ένα Ολοκληρωμένο Περιβάλλον Ανάπτυξης Εφαρμογών (IDE) που έχει ως σκοπό τη διδασκαλία του Αντικειμενοστραφούς προγραμματισμού με τη γλώσσα Java. Με το Greenfoot μπορεί να δημιουργηθεί ένας κόσμος μέσα στον οποίο αλληλεπιδρούν μορφές (actors) με τις οποίες μπορούν να προγραμματιστούν παιχνίδια, προσομοιώσεις και άλλα προγράμματα με γραφικά.

Ένας μαθητής Δευτεροβάθμιας θα πρέπει να αποκτήσει βασικές γνώσεις προγραμματισμού. Ο προγραμματισμός στο σχολείο είναι μια παγκόσμια τάση και παρά τις αντιρρήσεις που έχουν τεθεί πρόκειται για μια δεξιότητα που σίγουρα έχει κατακτήσει την θέση της στο σύνολο των δεξιοτήτων του 21ου αιώνα.

Στην ιστορία των υπολογιστών στην εκπαίδευση τα προγραμματιστικά περιβάλλοντα εισήχθησαν με τη Logo. Στα επόμενα χρόνια η διδασκαλία της Logo ατόνησε και θεωρήθηκε πιο σημαντική η καλλιέργεια δεξιοτήτων χρήσης υπολογιστή με έμφαση στις εφαρμογές γραφείου. Στις μέρες μας επιχειρείται μια αντίστροφη πορεία, με τον προγραμματισμό στον πυρήνα του Προγράμματος Σπουδών της Πληροφορικής.

Σήμερα η εποχή είναι διαφορετική. Οι εφαρμογές για κινητά τηλέφωνα και ταμπλέτες (mobile apps) βρίσκονται παντού και άνθρωποι που δεν σκέφτηκαν ποτέ να προγραμματίσουν, άρχιζαν να έχουν ιδέες για εφαρμογές και να ενδιαφέρονται για τον τρόπο με τον οποίο μπορούν να τις κάνουν πράξη.

Οι δύο αυτές τάσεις οδηγούν εταιρείες και οργανισμούς στην ανάπτυξη λογισμικών για την εκμάθηση του προγραμματισμού. Τα λογισμικά αυτά είναι πλέον πάρα πολλά, για παράδειγμα:

- Alice
- Greenfoot
- Scratch
- Kodu
- BYOB
- AgentSheets
- Microworlds Pro
- GameMaker
- Blockly
- StarLogo
- TurtleArt
- Squeak eToys

ενώ υπάρχουν και διαδικτυακές εφαρμογές:

<https://www.makegameswith.us/>

<https://groklearning.com/>

<https://www.khanacademy.org/cs>

<http://appinventor.mit.edu/explore/>

<http://www.crunchzilla.com/>

Από όλα αυτά τα περιβάλλοντα γιατί να επιλεγεί το Greenfoot; Για να απαντηθεί αυτό το ερώτημα θα πρέπει να δούμε τη συνέχεια της Διδασκαλίας του Προγραμματισμού στο Πρόγραμμα Σπουδών Πληροφορικής.

Στα Δημοτικά Σχολεία με ΕΑΕΠ διδάσκεται συνήθως Scratch στην Ε' και στην Στ' Δημοτικού, δεδομένου ότι είναι ένα λογισμικό εξελληνισμένο, δωρεάν και απευθύνεται

στις αντίστοιχες ηλικίες. Μια αναζήτηση<sup>1</sup> στο Σχολικό Δίκτυο παράγει πληθώρα αποτελεσμάτων. Το λογισμικό Microworlds Pro χρησιμοποιείται σε όλα τα Γυμνάσια αφού υπάρχει στο αντίστοιχο σχολικό εγχειρίδιο δημιουργώντας μια παράδοση που δύσκολα θα αλλάξει. Με το νέο πιλοτικό πρόγραμμα Σπουδών στο Γυμνάσιο ο προγραμματισμός είναι αντικείμενο και στις τρεις τάξεις του Γυμνασίου. Το θέμα είναι να επιλεγεί ένα λογισμικό που θα μπορεί να αποτελέσει την συνέχεια της ακολουθίας Scratch και Microworlds Pro.

Η βασική διδακτική επιλογή που έγινε από τους δημιουργούς του Greenfoot είναι η προσέγγιση "πρώτα τα αντικείμενα"<sup>2</sup>. Στην ουσία τόσο η Scratch όσο και το Microworlds Pro ακολουθώντας μια από τις αρχές της Logo, τα αντικείμενα σκέψης (objects to think with) έχουν επίσης σαν κεντρική ιδέα τα αντικείμενα. Και στο Greenfoot ο μαθητής προγραμματίζει την συμπεριφορά των αντικειμένων όπως έκανε και στα προηγούμενα περιβάλλοντα. Η σειρά Scratch Greenfoot μοιάζει πάρα πολύ με τη σειρά Alice Greenfoot που προτείνεται από την Oracle<sup>3</sup>.

Ο μαθητής προγραμματίζει σε Java. Δεν προγραμματίζει σε κάποια Java-like γλώσσα ή μια γλώσσα προγραμματισμού που είναι σχεδιασμένη αποκλειστικά για μαθητές. Προγραμματίζει σε μια επαγγελματική γλώσσα προγραμματισμού με την οποία μπορεί αργότερα να κατασκευάσει εφαρμογές υψηλών απαιτήσεων (οι εφαρμογές του Android προγραμματίζονται σε Java<sup>4</sup>).

Τέλος, το περιβάλλον παρέχεται δωρεάν ως λογισμικό ανοιχτού κώδικα και υποστηρίζεται από το Πανεπιστήμιο του Kent.

---

<sup>1</sup> Αναζητείστε στο Google: *Scratch Δημοτικό site:sch.gr*

<sup>2</sup> <http://blogs.kent.ac.uk/mik/docs/i-object/>

<sup>3</sup> <https://academy.oracle.com/oa-web-introcs-curriculum.html>

<sup>4</sup> <http://developer.android.com/tools/index.html>

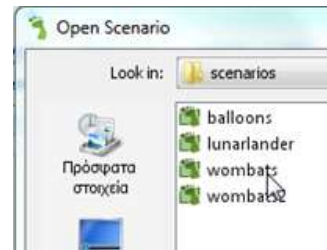
## 1. Εισαγωγή στο Greenfoot



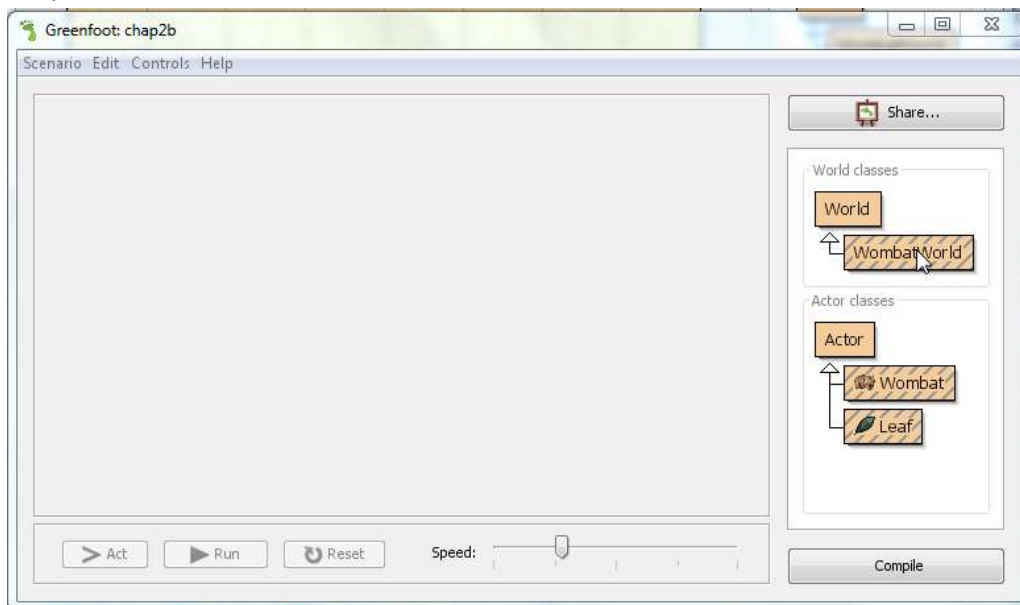
Από την σελίδα του Greenfoot<sup>5</sup> μπορούμε να κατεβάσουμε το λογισμικό (Download). Το λογισμικό μπορεί να εκτελεστεί σε οποιοδήποτε σύστημα μπορεί να εγκατασταθεί το Java Development Kit (JDK). Παρά το ότι το σύστημά σας μπορεί να υποστηρίζει Java, π.χ. να τρέχει εφαρμογές Java ή να τρέχει Java μέσω του

περιηγητή ιστού αυτό δεν σημαίνει ότι έχετε εγκατεστημένο το Java Development Kit. Ακολουθήστε λοιπόν ακριβώς τα βήματα που προτείνονται σε κάθε περίπτωση.

Στη συνέχεια ανοίξτε το έτοιμο σενάριο *wombats* που θα βρείτε στον φάκελο *scenarios* μέσα στον φάκελο εγκατάστασης του Greenfoot. Δηλαδή επιλέξτε «Scenario ➔ Open». Το Greenfoot θα σας ζητήσει να αποθηκεύσετε το έργο που μόλις ανοίξατε για να μπορεί να κάνει αλλαγές. Αποθηκεύστε το στον φάκελο Greenfoot που βρίσκεται μέσα στον φάκελο «Τα έγγραφά μου».



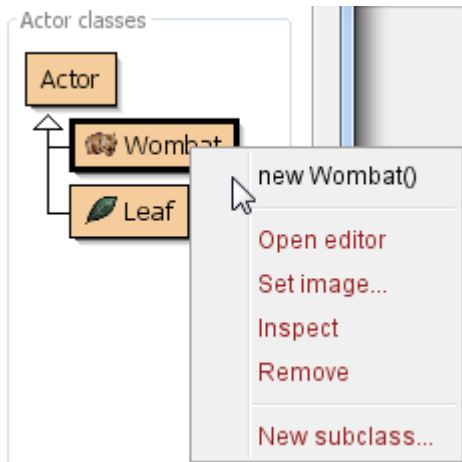
Πρόκειται για ένα σενάριο όπου το τρωκτικό *Wombat* κινείται μέσα στο πλέγμα του κόσμου *WombatWorld* και προσπαθεί να φάει τα φύλλα που πιθανόν υπάρχουν. Στο δεξί μέρος του περιβάλλοντος βλέπουμε την ιεραρχία των κλάσεων που αφορούν τον κόσμο (πάνω) και την ιεραρχία των Actor (κάτω). Κλάσεις είναι οι τύποι των αντικειμένων, π.χ. αν στον κόσμο υπάρχουν τρία ίδια φύλλα τότε αυτά ανήκουν στον ίδιο τύπο, στην ίδια κλάση αντικειμένων, την κλάση *Leaf*.



Κάθε λειτουργικό σενάριο θα πρέπει να έχει τουλάχιστον έναν κόσμο, και κάποιες μορφές (Actors). Στην περίπτωση του σεναρίου *wombats* οι μορφές είναι δύο, μια μορφή με όνομα *Wombat* (ένα είδος τρωκτικού) και μια μορφή με όνομα *Leaf* (φύλλο).

<sup>5</sup> <http://www.greenfoot.org/>

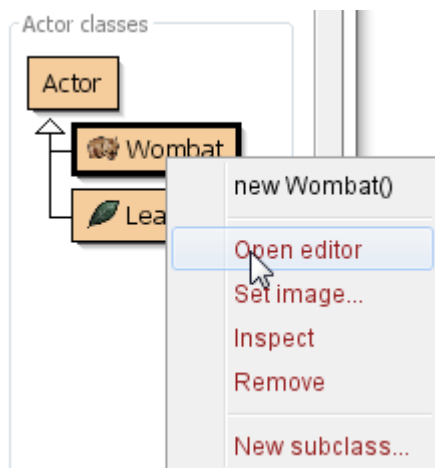
Για να εκτελέσουμε το σενάριο θα πρέπει πρώτα να κάνουμε **Compile** (μεταγλώττιση) με το κουμπί που βρίσκεται κάτω δεξιά. Αμέσως εμφανίζεται το πλέγμα του κόσμου στον οποίο θα κινούνται οι μορφές της εφαρμογής μας. Ο κόσμος είναι άδειος.



Μπορούμε να εισάγουμε ένα αντικείμενο τύπου Wombat. Αυτό γίνεται με δεξί κλικ πάνω στην κλάση του Wombat στην ιεραρχία των μορφών (Actor). Στη συνέχεια επιλέγουμε **new Wombat()**. Με το ποντίκι του υπολογιστή μπορούμε να το τοποθετήσουμε οπουδήποτε μέσα στον κόσμο.

Ακολουθούμε την ίδια διαδικασία με τα φύλλα (κλάση Leaf και επιλέγουμε **new Leaf()**). Μπορούμε να τοποθετήσουμε παραπάνω από ένα φύλλα στον κόσμο μας, επαναλαμβάνοντας την διαδικασία.

Για να εκτελέσουμε το σενάριο πατάμε το κουμπί **Run**. Βλέπουμε ότι το Wombat κινείται και μπορούμε να ρυθμίσουμε την ταχύτητα της εκτέλεσης με την μπάρα κύλισης **Speed**. Αν το Wombat βρει στον δρόμο του ένα φύλλο τότε το φύλλο εξαφανίζεται. Δοκιμάστε να βάλετε φύλλα σε αντίστοιχες θέσεις ώστε το wombat να τα φάει. Μπορείτε να εκτελείτε ένα βήμα κάθε φορά με το κουμπί **Act**, ουσιαστικά το κουμπί **Run** εκτελεί το **Act** πολλές φορές με την ταχύτητα **Speed**. Αν θέλετε να ξεκινήσετε από την αρχή πατήστε το κουμπί **Reset**.



Για να δούμε τον κώδικα για το Wombat μπορούμε να κάνουμε δεξί κλικ πάνω στην κλάση και να επιλέξουμε **Open editor**.

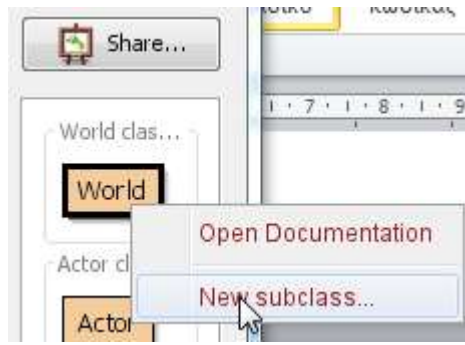
Φαίνεται ο παρακάτω κώδικας:

```
if(foundLeaf()) {
    eatLeaf();
}
else if(canMove()) {
    move();
}
else {
    turnLeft();
}
```

Ο κώδικας αυτός σημαίνει πως το wombat αν βρει φύλλο, θα το φάει αλλιώς αν μπορεί να πάει μπροστά θα πάει και αν δεν μπορεί να πάει μπροστά θα στρίψει αριστερά. Υπάρχει δηλαδή αντιστοιχία μεταξύ του κώδικα και της συμπεριφοράς του Wombat. Ο κώδικας αυτός βρίσκεται «μέσα» στις αγκύλες που ακολουθούν το **public void act()**. Οτιδήποτε υπάρχει μέσα στο **act()** εκτελείται όταν πατηθούν τα κουμπιά **Act** ή **Run**.

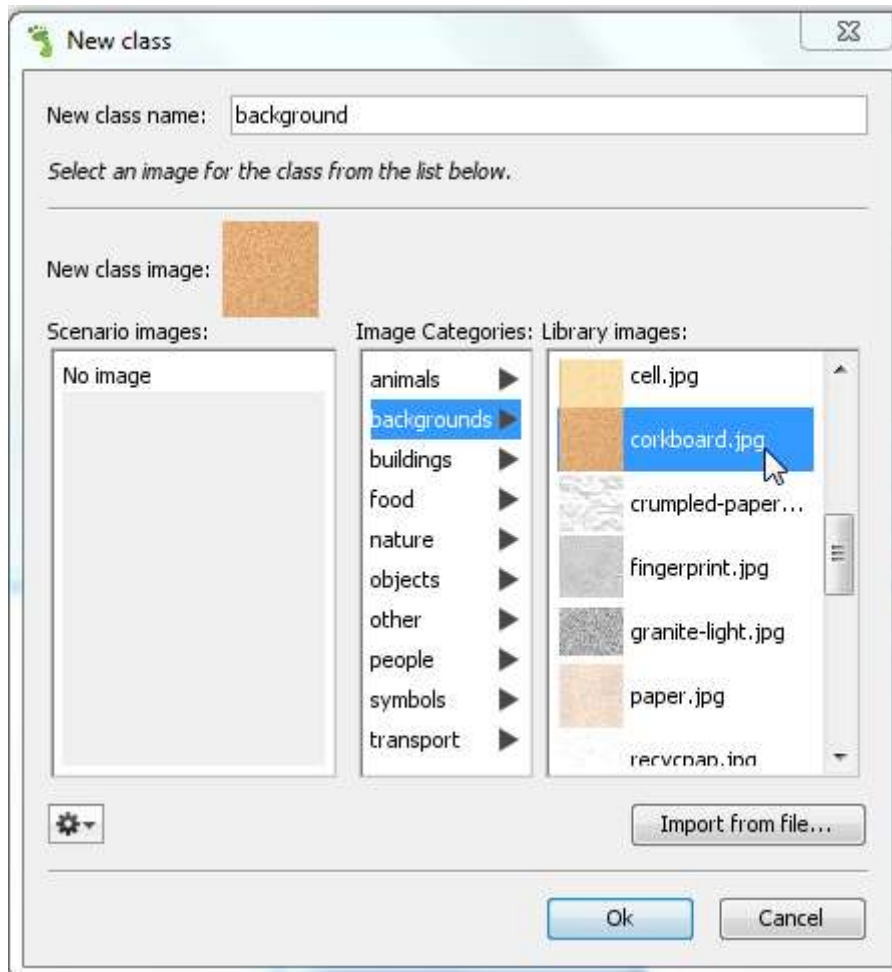
Στα επόμενα κεφάλαια θα βάλουμε τις δικές μας εντολές μέσα στην Act και θα φτιάξουμε τα δικά μας έργα.

## 2. Κίνηση!



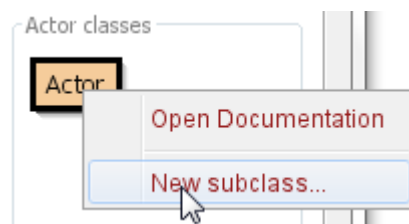
Σε κάθε έργο που ξεκινάμε πρέπει να δημιουργήσουμε μια υποκλάση της κλάσης World. Αυτό γίνεται κάνοντας δεξί κλικ στο εικονίδιο World και επιλέγοντας **New subclass**. Τη νέα υποκλάση μπορούμε να την ονομάσουμε background και να διαλέξουμε μια εικόνα για την αναπαράσταση του.

Το Greenfoot έχει εγκαταστήσει κάποια υπόβαθρα τα οποία μπορούμε να χρησιμοποιήσουμε στις εφαρμογές μας και βρίσκονται στον φάκελο **backgrounds**. Δοκιμάστε όποιο θέλετε από αυτά. Μπορείτε να χρησιμοποιήσετε και εικόνες από τον υπολογιστή σας.



Σε αυτό το έργο θα εισάγουμε μια μορφή (Actor) και θα την κάνουμε να κινείται μέσα στο πλέγμα του κόσμου που δημιουργήσαμε. Για να δημιουργήσουμε μια νέα μορφή κάνουμε δεξί κλικ στο εικονίδιο Actor και μετά **New subclass**.

Το Greenfoot έχει εγκατεστημένες πολλές εικόνες που μπορούν να χρησιμοποιηθούν σε έργα, χωρισμένες σε κατηγορίες. Σε αυτό το έργο χρησιμοποιήστε κάποια εικόνα από την κατηγορία **Animals** και δώστε στην υποκλάση ένα αντίστοιχο όνομα.







Παρά το ότι στην ιεραρχία των μορφών εμφανίστηκε το wombat, δεν εμφανίστηκε κανένα αντικείμενο μέσα στον κόσμο. Για να εισάγουμε αντικείμενα θα κάνουμε δεξί κλικ στο εικονίδιο του **wombat** και **new wombat()**, αφού πρώτα κάνουμε **Compile** (Μεταγλώττιση). Με το ποντίκι του υπολογιστή μπορούμε να βάλουμε το αντικείμενό μας όπου θέλουμε μέσα στο πλέγμα του κόσμου.

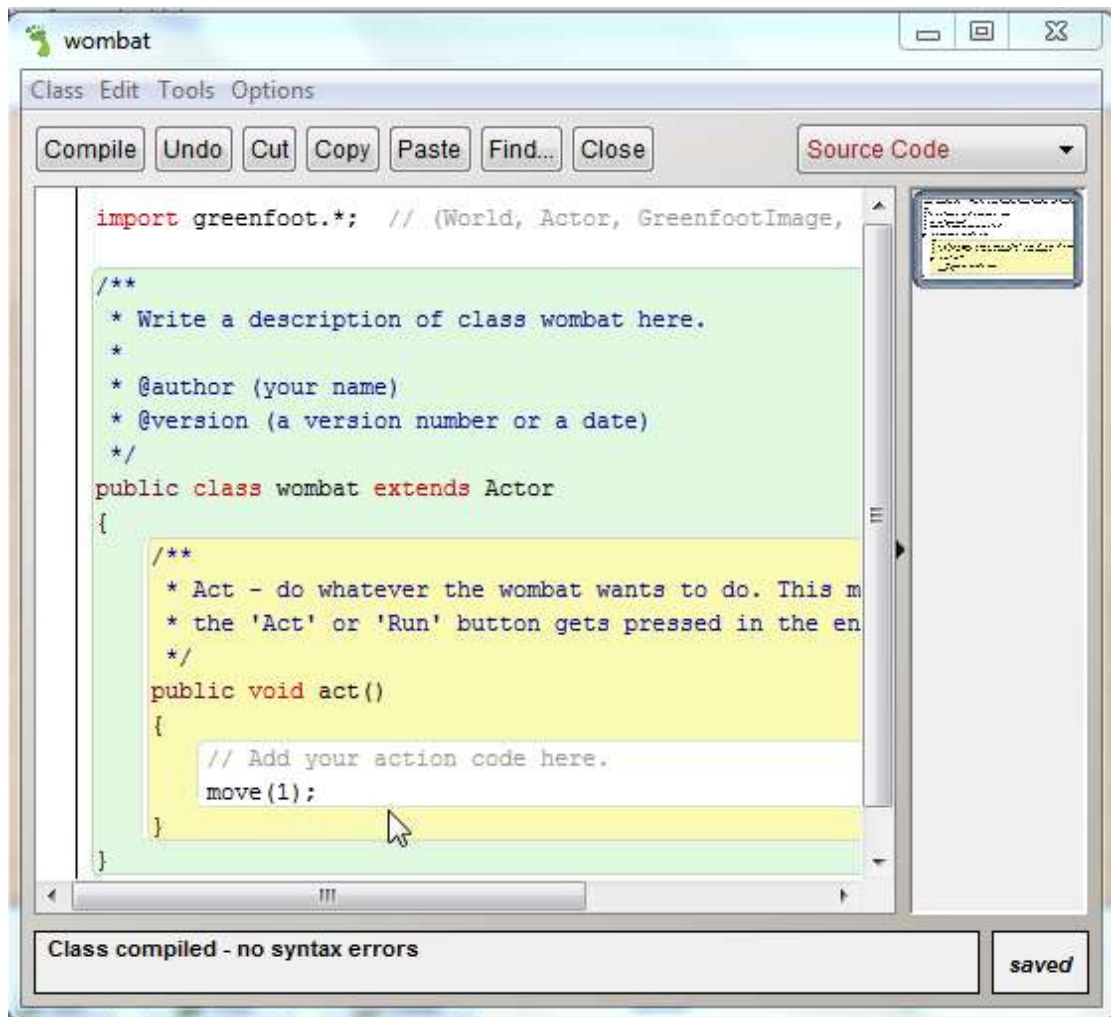
Αν πατήσουμε **Run** ή **Act** τίποτε δεν θα γίνει και αυτό γιατί δεν έχουμε προγραμματίσει το wombat να κάνει κάτι.

Για να το προγραμματίσουμε κάνουμε δεξί κλικ στο εικονίδιο του **wombat** και **Open editor** (Άνοιγμα επεξεργαστή κειμένου). Το Greenfoot δημιουργεί κώδικα για τη νέα κλάση χωρίς να υπάρχει κάτι μέσα στην μέθοδο **act**.

Για να προγραμματίσουμε το wombat να κινείται μέσα στον κόσμο μπορούμε να βάλουμε στην μέθοδο **act** την εντολή **move(1)**.

### Κώδικας 3.1:

```
public void act()
{
    // Add your action code here.
    move(1);
}
```



### Βίντεο



<https://www.youtube.com/watch?v=9uVEX9Tntac>

### Επέκταση 3.1:



Αλλάξτε την παράμετρο της εντολής `move` από 1 σε κάποιο μεγαλύτερο άριθμο (π.χ. 3 ή και ακόμη μεγαλύτερο). Ποιο χαρακτηριστικό της κίνησης αλλάζει;

### Ερώτηση 3.1.

Τι σημαίνει η εντολή `move(1)`;

1. Κίνηση κατά ένα μήκος σώματος προς τα πάνω.
2. Κίνηση κατά ένα εικονοστοιχείο προς τα πάνω.
3. Κίνηση κατά ένα εικονοστοιχείο (pixel) προς την κατεύθυνση της μορφής.
4. Κίνηση κατά ένα εικονοστοιχείο προς τα αριστερά.



### Ερώτηση 3.2.

Ποιος από τους παρακάτω κώδικες είναι σωστός;

A.

```
public class wombat extends Actor
public void act()
{
}
move(1);
}
```

B.

```
public class wombat extends Actor
move(1);
public void act()
{
}
}
```

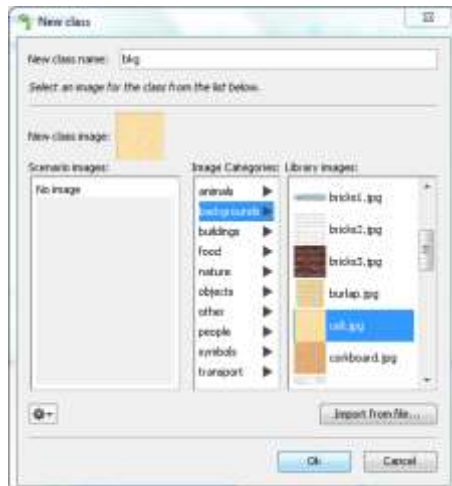
C.

```
public class wombat extends Actor
public void run()
{
move(1);
}
}
```

D.

```
public class wombat extends Actor
public void act()
{
move(1);
}
}
```

### 3. Τυχαία κίνηση



Στο κεφάλαιο αυτό μια μορφή (Actor) του προγράμματός μας θα κινείται με τυχαίο τρόπο μέσα στο υπόβαθρο. Δημιουργήστε ένα νέο υπόβαθρο (background) και χρησιμοποιήστε την εικόνα `cell.jpg` όπως φαίνεται στο παρακάτω σχήμα

Η εικόνα `cell.jpg` έχει μέγεθος 60x60 εικονοστοιχεία (pixel) και με αυτή μπορούμε να δημιουργήσουμε ένα πλέγμα με μεγάλα τετράγωνα.

Ανοίξτε τον editor για το υπόβαθρο (εντολή «Open Editor») και αλλάξτε τις παραμέτρους της εντολής `super`. Η εντολή `super` δημιουργεί το πλέγμα στο οποίο κινούνται οι μορφές.

Η σύνταξη της εντολής αυτής είναι η εξής:

**super** (αριθ\_κελιών\_οριζόντια, αριθ\_κελιών\_κάθετα, μέγεθος)

Αν το μέγεθος κελιού είναι 1 όπως μέχρι τώρα τότε τα αντικείμενα μπορούν να κινηθούν ελεύθερα μέσα στο πλέγμα.

Επειδή χρησιμοποιούμε την εικόνα `cell.jpg` θα αλλάξουμε το μέγεθος του κελιού σε 60. Για αυτό τον λόγο θα χρησιμοποιήσουμε 8 κελιά στον οριζόντιο άξονα και 8 κελιά στον κάθετο άξονα. Θα πρέπει να αλλάξετε και το κείμενο που βρίσκεται στα σχόλια ώστε να συμφωνεί με την αλλαγή που κάνατε.

#### Κώδικας 4.1:

```
super (8, 8, 60);
```

Τώρα μπορούμε να εισάγουμε ένα αντικείμενο μέσα στο πλέγμα. Δημιουργήστε μια νέα υποκλάση της κλάσης `Actor` με την εικόνα ενός ποντικού, ονομάστε τη `mouse`.

Έχουμε μάθει με ποιον τρόπο μπορεί μια μορφή (Actor) να κινηθεί μέσα στο πλέγμα. Δεδομένου ότι το πλέγμα μας έχει μέγεθος 8x8 κελιά θα χρησιμοποιήσουμε μικρές τιμές για την παράμετρο της εντολής `move`. Δοκιμάστε την εντολή `move(1)`; Θα δείτε ότι το ποντίκι κινείται από κελί σε κελί και δεν κάνει μικρές κινήσεις όπως στα έργα που έχετε υλοποιήσει μέχρι τώρα.

Επειδή το ποντίκι δεν στρίβει, σύντομα φτάνει στο τελευταίο κελί του πλέγματος και σταματάει. Μπορούμε να κάνουμε το ποντίκι να στρίβει με την εντολή `turn`.

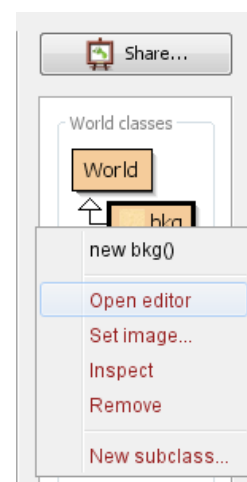
Αν χρησιμοποιήσουμε τις εντολές

```
move (1);  
turn (90);
```

το ποντίκι καλύπτει μόνο μια μικρή περιοχή του πλέγματος.

Μπορούμε να κάνουμε το ποντίκι να μην στρίβει κάθε φορά, αλλά να στρίβει μόνο τις μισές φορές και μάλιστα με τυχαίο τρόπο. Για να το καταφέρουμε θα χρειαστούμε την εντολή:

```
Greenfoot.getRandomNumber (όριο)
```



Η εντολή αυτή επιστρέφει έναν τυχαίο αριθμό από το 0 μέχρι το όριο. Το όριο είναι ένας ακέραιος αριθμός (int). Μπορείτε να δείτε πώς λειτουργεί η εντολή αυτή χρησιμοποιώντας την εντολή

```
System.out.println
```

#### Κώδικας 4.2:

```
move (1) ;  
System.out.println (Greenfoot.getRandomNumber (100) ) ;
```

Αν εκτελέσετε τον παραπάνω κώδικα εμφανίζεται ένα νέο παράθυρο με τίτλο «Greenfoot: Terminal Window». Κάθε φορά που πατάτε Act μια νέα τυχαία τιμή εμφανίζεται σε αυτό το παράθυρο. Η τιμή αυτή είναι το αποτέλεσμα της μεθόδου **Greenfoot.getRandomNumber**. Επειδή το όριο είναι 100 οι αριθμοί αυτοί μπορεί να είναι από 0 έως 99.

Μπορούμε να προγραμματίσουμε το ποντίκι να στρίβει μόνο αν ο αριθμός αυτός είναι μεγαλύτερος από το 50. Για να το κάνουμε αυτό θα χρειαστούμε την εντολή **if (εάν)**.

#### Κώδικας 4.3:

```
move (1) ;  
if (Greenfoot.getRandomNumber (100) > 50)  
{  
    turn (90) ;  
}
```

Το ποντίκι στρίβει μόνο αν ο τυχαίος αριθμός είναι μεγαλύτερος του 50. Όμως αυτή δεν ξέρουμε πότε θα συμβεί δεδομένου ότι η μέθοδος **Greenfoot.getRandomNumber** παράγει έναν τυχαίο αριθμό κάθε φορά. Έτσι το ποντίκι κάνει μια τυχαία κίνηση μέσα στο πλέγμα.

#### Βίντεο



<https://www.youtube.com/watch?v=JBth1hrjTM8>

#### Επέκταση 4.1:



Μπορείτε να προγραμματίσετε το ποντίκι ώστε να μην στρίβει τόσο συχνά; Ποια παράμετρο του προγράμματός σας θα αλλάξετε;

#### Επέκταση 4.2:



Μπορείτε να βάλετε παραπάνω από ένα αντικείμενα να κινούνται με διαφορετικούς τρόπους (με διαφορετικές παραμέτρους στις τιμές move, turn αλλά και με διαφορετική πιθανότητα περιστροφής;

#### Ερώτηση 4.1:

Πώς μπορούμε να δημιουργήσουμε έναν κόσμο με 4 κελιά στον οριζόντιο άξονα και 8 κελιά στον κάθετο άξονα.

1. `super(μέγεθος_κελιού, 4, 8);`
2. `world(4, 8, μέγεθος_κελιού);`
3. `super(4, 8, μέγεθος_κελιού);`
4. `newworld(4, 8, μέγεθος_κελιού);`

#### Ερώτηση 4.2:

Πώς μπορούμε να προσομοιώσουμε ένα ζάρι με το Greenfoot;

1. `Greenfoot.getRandomNumber(6)+1;`
2. `Greenfoot.getRandomNumber(7);`
3. `Greenfoot.getRandomNumber(6);`
4. `Greenfoot.getRandom(6);`

#### Ερώτηση 4.3.

Σε ποιον από τους παρακάτω κώδικες το ποντίκι κινείται μπροστά σε κάθε act.

A

```
move(1);  
if (Greenfoot.getRandomNumber(100)>50)  
{  
turn(90);  
}
```

B

```
if (Greenfoot.getRandomNumber(100)>50)  
{  
turn(90);  
move(1);  
}
```

Γ

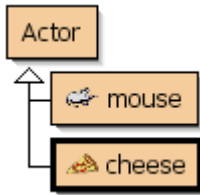
```
if (Greenfoot.getRandomNumber(100)>50)  
{  
move(1);  
}  
turn(90);
```

Δ

```
turn(90);  
if (Greenfoot.getRandomNumber(100)>50)  
{  
move(1);  
}
```

## 5. Αλληλεπίδραση αντικειμένων

Στο Κεφάλαιο 4 προγραμματίσαμε ένα ποντίκι ώστε να κινείται με τυχαίο τρόπο. Σε αυτό το κεφάλαιο θα προσθέσουμε ένα τυράκι και θα πρέπει όταν το ποντίκι ακουμπήσει το τυράκι το τυράκι να εξαφανίζεται (τρώγεται).



Εισάγετε μια νέα μορφή (*Actor*) με όνομα **cheese** και εικόνα *pizza\_cheese.png*. Η μορφή αυτή δεν θα έχει κάποιο πρόγραμμα θα προγραμματίσουμε τη μορφή **mouse** να την εξαφανίζει όταν την ακουμπήσει. Για να καταλάβει η μορφή **mouse** αν ακουμπάει μια άλλη μορφή υπάρχει η μέθοδος **isTouching()**. Για να περιορίσουμε την εντολή ώστε να ανιχνεύει αν αυτό που ακούμπησε ήταν αντικείμενο της

κλάσης **cheese** μπορούμε να χρησιμοποιήσουμε τη μέθοδο ως εξής:

```
isTouching(cheese.class)
```

### Κώδικας 5.1:

```
//Random movement
move(1);
if (Greenfoot.getRandomNumber(100)>50)
{
    turn(90);
}
//Check if it is touching cheese
if (isTouching(cheese.class))
{
    System.out.println("Touching");
}
```

Θα δείτε ότι την πρώτη φορά που θα ακουμπήσει η μορφή **mouse** την μορφή **cheese** θα εμφανιστεί ένα παράθυρο με τη λέξη *Touching*, από εκείνη τη στιγμή και μετά στο παράθυρο θα εμφανίζεται η λέξη κάθε φορά που το ποντίκι ακουμπάει το τυράκι.



Αντί να εμφανίζεται η λέξη *Touching* το ποντίκι θα έπρεπε να εξαφανίζει το τυράκι. Αυτό μπορεί να γίνει με την εντολή **removeTouching** που επίσης θα χρησιμοποιηθεί με όρισμα **cheese.class**.

### Κώδικας 5.2:

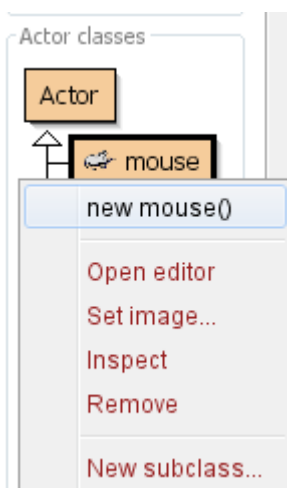
```
//Random movement
move(1);
if (Greenfoot.getRandomNumber(100)>50)
{
    turn(90);
}
//Check if it is touching cheese
if (isTouching(cheese.class))
{
    removeTouching(cheese.class);
}
```

Μπορούμε να χρησιμοποιήσουμε δικές μας μεθόδους για τις ομάδες των εντολών που επιτελούν μια συγκεκριμένη λειτουργία. Οι μέθοδοι εφόσον ασχολούνται μόνο με την κίνηση και δεν υπολογίζουν κάποιο αποτέλεσμα μπορούν να μην επιστρέφουν τίποτε. Έτσι δηλώνονται με **void** ονομα\_συνάρτησης().

Οι νέες μέθοδοι θα δηλωθούν έξω από το σώμα της **act()**. Είναι η πρώτη φορά που θα γράψουμε κώδικα έξω από τη μέθοδο **act** οπότε θα χρειαστεί ιδιαίτερη προσοχή.

### Κώδικας 5.3:

```
public void act()
{
    // Add your action code here.
    randomMove();
    touchingCheese();
}
void randomMove() {
    move(1);
    if (Greenfoot.getRandomNumber(100)>50)
    {
        turn(90);
    }
}
void touchingCheese() {
    if (isTouching(cheese.class))
    {
        removeTouching(cheese.class);
    }
}
```



Για να δημιουργήσουμε αντικείμενα μιας κλάσης, όπως θυμάστε, πατάμε δεξί κλικ στο εικονίδιο της κλάσης και στη συνέχεια **new** **όνομα\_κλάσης()**. Έτσι, εμφανίζεται ένα αντικείμενο της επιλεγμένης κλάσης και το τοποθετούμε πάνω στο πλέγμα. Αυτό μπορεί να γίνει και με την εντολή **addObject()**, με την εντολή αυτή δεν θα χρειάζεται να βάζουμε τα αντικείμενα χειροκίνητα κάθε φορά αλλά θα εμφανίζονται αυτόματα κάθε φορά που πατάμε **Compile**. Η εντολή **addObject** εκτελείται μέσα στον constructor του world (δεξί κλικ στην υποκλάση του world και επιλέγετε **Open Editor**).



Η εντολή **addObject()** συντάσσεται ως εξής:

```
addObject (new όνομα_κλάσης ( ) , θέση_x , θέση_y ) ;
```

Με το **new** όνομα\_κλάσης ( ) δημιουργείται ένα αντικείμενο αυτής της κλάσης και με την εντολή **addObject** το αντικείμενο αυτό εισάγεται σε συγκεκριμένη θέση στο πλέγμα.

Για παράδειγμα, μπορούμε να κάνουμε το ποντίκι να εμφανίζεται στη μέση της οθόνης κάθε φορά.

#### Κώδικας 5.4:

```
super ( 8 , 8 , 60 ) ;  
addObject ( new mouse ( ) , 4 , 4 ) ;
```

Ο κώδικας αυτός θα εισαχθεί στον **constructor** της υποκλάσης **bkg** που έχουμε δημιουργήσει

#### Βίντεο



<https://www.youtube.com/watch?v=a22qCCd7b6g>

#### Επέκταση 5.1:



Μπορείτε να εισάγετε αυτόματα ένα τυράκι, σε συγκεκριμένη θέση; Π.χ. στην θέση 1,6.

#### Επέκταση 5.2:



Μπορείτε να εισάγετε αυτόματα τρία τυράκια, σε τυχαίες θέσεις; Θυμηθείτε τη μέθοδο **Greenfoot.getRandomNumber()**.

#### Επέκταση 5.3:



Μπορείτε να βάλετε τις εντολές **addNewObject()** που χρησιμοποιήσατε σε μια νέα μέθοδο με όνομα **addObjects**; Έτσι ο **constructor** της **bkg** θα έχει μόνο δύο εντολές, την **super** και την **addObjects**.

#### Επέκταση 5.4:



Προσθέστε το αντικείμενο **lemur** με εικόνα **lemur.png**. Προγραμματίστε το να κινείται και αυτό με τυχαίο τρόπο και όποτε αγγίζει το ποντίκι να το τρώει.

#### Ερώτηση 5.1:

Πώς μπορούμε να ελέγξουμε αν μια μορφή αγγίζει μια άλλη μορφή κλάσης **wombat**.

1. `if (isTouching(wombat))`
2. `if (Touching(wombat))`
3. `if (isTouching(wombat.class))`
4. `if (removeTouching(wombat.class))`

### Ερώτηση 5.2:

Τι σημαίνει η εντολή `removeTouching(wombat.class)`;

1. Αφαίρεσε από τον κόσμο το αντικείμενο που εκτελεί την εντολή `removeTouching`.
2. Αφαίρεσε από τον κόσμο όλα τα αντικείμενα της κλάσης `wombat`.
3. Αφαίρεσε από τον κόσμο τα αντικείμενα της κλάσης `wombat` που ακουμπούν το αντικείμενο που εκτελεί την `removeTouching`.
4. Αφαίρεσε από τον κόσμο όλα τα αντικείμενα που εκείνη την ώρα ακουμπάνε σε αντικείμενα κλάσης `wombat`.

### Ερώτηση 5.3:

Ποιος από τους παρακάτω κώδικες μπορεί να μεταγλωττιστεί (compile);

A.

```
void act() {
    //Υλοποίηση της act
    void move();
    void eat();
}
void move() {
    //Υλοποίηση της move
}
void eat() {
    //Υλοποίηση της eat
}
```

B.

```
void act() {
    //Υλοποίηση της act
    move();
    eat();
}
void move() {
    //Υλοποίηση της move
}
void eat() {
    //Υλοποίηση της eat
}
```

Γ.

```
void act() {
    //Υλοποίηση της act
    move();
    eat();
}
void move() {
    //Υλοποίηση της move

    void eat() {
        //Υλοποίηση της eat
    }
}
```

Δ.

```
void act() {
    //Υλοποίηση της act
    move();
    eat();

    void move() {
        //Υλοποίηση της move
    }
    void eat() {
        //Υλοποίηση της eat
    }
}
```

## 6. Το πρώτο παιχνίδι

Ένα πρώτο βήμα για να γίνει η εφαρμογή που κατασκευάσατε στην επέκταση 5.4 παιχνίδι, είναι να κινείται το ποντίκι με τα πλήκτρα του πληκτρολογίου. Η εντολή που το επιτρέπει αυτό είναι η **Greenfoot.isKeyDown()**. Η φράση **isKeyDown** σημαίνει «Είναι το πλήκτρο πατημένο» χωρίς να αναφέρεται σε κάποιο πλήκτρο συγκεκριμένα. Το πλήκτρο δηλώνεται μέσα στην παρένθεση και μπορεί να είναι οποιοδήποτε πλήκτρο του πληκτρολογίου. Για να χρησιμοποιήσουμε τα βελάκια μέσα στην παρένθεση θα βάλουμε τις λέξεις ("up","down","left","right"). Έτσι αντικαθιστούμε τη μέθοδο **randomMove** με τη μέθοδο **checkKeyboard** μέσα στην **act** της κλάσης **mouse**.

### Κώδικας 6.1:

```
public void act()
{
    checkKeyboard();
    touchingCheese();
}
void checkKeyboard() {
    if (Greenfoot.isKeyDown("left"))
    {
        turn(-90);
    }
    if (Greenfoot.isKeyDown("right"))
    {
        turn(90);
    }
    if (Greenfoot.isKeyDown("up")) {
        move(1);
    }
}
```

Η μέθοδος  
**touchingCheese**  
δεν αλλάζει.

Παρατηρείτε ότι με το αριστερό και το δεξί πλήκτρο το ποντίκι στρίβει, ενώ με το πάνω προχωράει προς την κατεύθυνση που έχει τη δεδομένη χρονική στιγμή.

Μπορούμε να σταματήσουμε το παιχνίδι όταν το αντικείμενο **lemur** ακουμπήσει το ποντίκι. Αυτό μπορεί να γίνει με την εντολή **Greenfoot.stop()**. Η οποία θα πρέπει να εκτελεστεί όταν το **lemur** ακουμπήσει το **mouse**, και κατά συνέπεια πρέπει να μπει στον κώδικα του **lemur**.

### Κώδικας 6.2:

```
void touchingMouse() {
    if (isTouching(mouse.class))
    {
        removeTouching(mouse.class);
        Greenfoot.stop();
    }
}
```

Το παιχνίδι θα πρέπει να σταματάει και στην περίπτωση που το ποντίκι φάει όλα τα τυράκια. Με κάποιο τρόπο θα πρέπει να ξέρουμε πόσα τυράκια έχουν φαγωθεί (το σκορ). Δεν είναι ανάγκη όλες οι μορφές να ξέρουν το σκορ, αρκεί να το υπολογίζει το αντικείμενο

της κλάσης `mouse`. Αρχικά το σκορ είναι 0 καθώς το ποντίκι τρώει τυράκια, το σκορ μπορεί να αυξάνεται. Τέλος το σκορ είναι ακέραιος αριθμός (δεν έχει δεκαδικά ψηφία). Έτσι δημιουργούμε μια μεταβλητή ακεραίου με όνομα `score` και αρχική τιμή 0 (εντολή `int score = 0`). Την εντολή αυτή τη βάζουμε πάνω από τις συναρτήσεις ώστε όλες οι συναρτήσεις να έχουν πρόσβαση στη μεταβλητή `score`.

Κάθε φορά που το ποντίκι τρώει ένα τυράκι η τιμή `score` πρέπει να αυξηθεί. Έτσι στην τιμή `score` προσθέτουμε το 1 (`score + 1`). Αυτή πρέπει να είναι η νέα τιμή της μεταβλητής `score` (`score = score + 1`). Η διαδικασία αυτή γίνεται όταν εξαφανίζουμε ένα τυράκι.

#### Κώδικας 6.3:

```
int score = 0;
public void act()
{
    checkKeyboard();
    touchingCheese();
}
void touchingCheese() {
    if (isTouching(cheese.class))
    {
        removeTouching(cheese.class);
        score = score + 1;
        System.out.println(score);
    }
}
```

Η μέθοδος `checkKeyboard` δεν αλλάζει.

Κάθε φορά που το ποντίκι «τρώει» ένα τυράκι, εμφανίζεται ένα παράθυρο με το σκορ. Ωστόσο, δεν είναι αυτή η λειτουργικότητα που επιθυμούμε. Μπορούμε να κάνουμε το παιχνίδι να τελειώσει όταν το `score` γίνει 3. Χρησιμοποιούμε την εντολή `Greenfoot.stop()` και ελέγχουμε αν η τιμή της μεταβλητής `score` είναι 3 χρησιμοποιώντας τον τελεστή `==`.

#### Κώδικας 6.4:

```
void touchingCheese() {
    if (isTouching(cheese.class))
    {
        removeTouching(cheese.class);
        score = score + 1;
        if (score == 3)
        {
            Greenfoot.stop();
        }
    }
}
```

#### Επέκταση 6.1:



Μπορείτε να εισάγετε κατάλληλες εντολές ώστε το ποντίκι να κάνει μεταβολή όταν πατηθεί το κάτω πλήκτρο του πληκτρολογίου.

#### Επέκταση 6.2:



Τι θα κάνετε για να βάλετε περισσότερα τυράκια στο παιχνίδι; Τι θα πρέπει να αλλάξετε στον κώδικα της κλάσης `mouse`;

### Ερώτηση 6.1:

Πώς μπορούμε να ανιχνεύσουμε το πάτημα του πλήκτρου "Κάτω Βελάκι";

1. `Greenfoot.isKeyDown("down");`
2. `isKeyDown("down");`
3. `Greenfoot.isKey("down");`
4. `Greenfoot.is("keydown");`

### Ερώτηση 6.2:

Πώς σταματάμε την εκτέλεση ενός προγράμματος στο Greenfoot;

1. `stop();`
2. `Greenfoot.stop();`
3. `this.stop();`
4. `Greenfoot.halt();`

### Ερώτηση 6.3:

Τι κάνει η εντολή `int score = 0;`

1. Δημιουργεί μια νέα μεταβλητή με όνομα `score`, τιμή μηδέν και τύπο ακεραίου.
2. Δημιουργεί μια νέα μεταβλητή με όνομα `int`, τιμή μηδέν και τύπο `score`.
3. Δημιουργεί ένα νέο αντικείμενο κλάσης `score`.
4. Δημιουργεί μια νέα μεταβλητή με όνομα `int`, τιμή μηδέν και τύπο `score`.

### Ερώτηση 6.4:

Ποιός από τους παρακάτω κώδικες υλοποιεί τις ζωές μιας μορφής; Η μορφή αυτή χάνει μια ζωή όταν ακουμπήσει αντικείμενα κλάσης `shark...`

**A**

```
lives = 3;
public void act(){
    if (isTouching(shark.class)){
        lives = lives - 1;
    }
}
```

**B**

```
int lives = 3;
public void act(){
    if (isTouching(shark.class)){
        lives = - 1;
    }
}
```

**Γ**

```
int lives = 3;
public void act(){
    if (isTouching(shark.class)){
        lives = lives - 1;
    }
}
```

**Δ**

```
int lives = 3;
public void act(){
    if (isTouching(shark.class)){
    }
    lives = lives - 1;
}
```

### Ερώτηση 6.5:

Ποιος από τους παρακάτω κώδικες σταματάει το παιχνίδι όταν έχουν μηδενιστεί οι ζωές (μεταβλητή lives).

A.

```
if (lives = 0){  
Greenfoot.stop();  
}
```

B.

```
if (lives == 0){  
Greenfoot.stop();  
}
```

Γ.

```
if (lives is 0){  
Greenfoot.stop();  
}
```

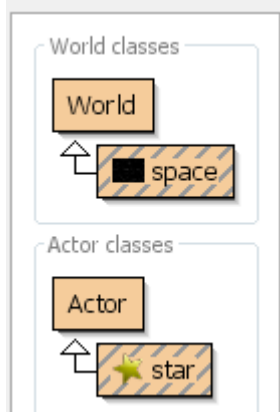
Δ.

```
if (lives equals 0){  
Greenfoot.stop();  
}
```



## 7. Κλικ!

Στα επόμενα κεφάλαια θα χτίσουμε μια εφαρμογή βήμα-βήμα. Ξεκινώντας από ένα παιχνίδι που ο χρήστης πατάει με το ποντίκι του ένα αστέρι, θα φτάσουμε σε μια εφαρμογή που μπορεί να απασχολήσει τον χρήστη για αρκετό χρόνο. Πρόκειται για την υλοποίηση σε Greenfoot ενός βιντεομαθήματος<sup>6</sup>.



Αρχικά δημιουργούμε έναν κόσμο `space` και την υποκλάση του `Actor` με όνομα `star` και αντίστοιχη εικόνα. Επειδή δεν υπάρχει προεγκατεστημένη εικόνα αστεριού στο περιβάλλον του Greenfoot θα χρησιμοποιήσετε μία εικόνα από το διαδίκτυο. Δώστε ιδιαίτερη σημασία στα πνευματικά δικαιώματα. Αν θέλετε να χρησιμοποιήσετε την ίδια εικόνα που βλέπετε μπορείτε να την βρείτε στη διεύθυνση <http://www.clker.com/clipart-24889.html>.

Το προκαθορισμένο μέγεθος του κόσμου είναι 600x400 pixels (εικονοστοιχεία) το οποίο καλύπτει τις ανάγκες μας για αυτό το παιχνίδι. Τώρα πρέπει να εισάγετε το αστέρι στον κόσμο με την εντολή `addObject`.

### Κώδικας 7.1:

```
addObject(new star(), 300, 200);
```

Το επόμενο βήμα είναι να κάνουμε το αστέρι να εξαφανίζεται όταν γίνει κλικ πάνω του. Αυτό γίνεται με τον παρακάτω κώδικα.

### Κώδικας 7.2:

```
if (Greenfoot.mouseClicked(this))
{
    getWorld().removeObject(this);
}
```

Η λέξη `this` αναφέρεται στο αντικείμενο που εκτελεί τον κώδικα. Δηλαδή στη συγκεκριμένη περίπτωση αναφέρεται στο αντικείμενο του αστεριού που υπάρχει στον κόσμο. Το Greenfoot έχει υλοποιήσει μια μέθοδο `mouseClicked` η οποία μπορεί να ελέγξει αν έχει γίνει κλικ πάνω στο αντικείμενο. Το ίδιο συμβαίνει και με την μέθοδο `removeObject(this)` η οποία αφαιρεί από τον κόσμο το συγκεκριμένο αντικείμενο. Η μέθοδος `removeObject` ανήκει σε αντικείμενα της κλάσης `World`. Το κάθε αντικείμενο μπορεί να αναγνωρίσει τον κόσμο στον οποίο βρίσκεται με την εντολή `getWorld()`.

Το αντικείμενο εξαφανίζεται μόνο αν έχει πατηθεί το `Run`, αν δεν έχει πατηθεί το `Run` τότε δεν εκτελούμε το πρόγραμμα αλλά προγραμματίζουμε. Σε αυτή την περίπτωση όταν το ποντίκι του υπολογιστή βρίσκεται πάνω από το αστέρι εμφανίζεται το χεράκι και όχι ο δείκτης.

### Επέκταση 7.1:



Μπορείτε να εισάγετε ένα δεύτερο αντικείμενο που να εξαφανίζεται όταν πατηθεί με το ποντίκι;

<sup>6</sup> <http://www.youtube.com/watch?v=vve-0438LxA>

## Επέκταση 7.2:



Μπορείτε να προγραμματίσετε το αστέρι ώστε να πηγαίνει σε μια τυχαία θέση αντί να εξαφανίζεται; (Χρησιμοποιήστε την εντολή setLocation).

## Ερώτηση 7.1:

Τι κάνει η εντολή `addObject(new star(),300,200)`;

1. Εισάγει την κλάση `star` στον κόσμο.
2. Εισάγει ένα νέο αντικείμενο της κλάσης `star` στον κόσμο σε τυχαία θέση.
3. Εισάγει ένα νέο αντικείμενο της κλάσης `star` στον κόσμο σε συγκεκριμένη θέση.
4. Εισάγει ένα νέο αντικείμενο της κλάσης `object` στον κόσμο στη θέση που βρίσκεται αντικείμενο της κλάσης `star`.

## Ερώτηση 7.2:

Ποιος από τους παρακάτω κώδικες που βρίσκεται μέσα στην `act` κάποιου `Actor` είναι σωστός;

1. `this.removeObject();`
2. `getWorld().removeThis();`
3. `getWorld().removeObject(this);`
4. `World().removeObject(this);`

## Ερώτηση 7.3:

Ποιος από τους παρακάτω κώδικες ανιχνεύει αν έχει γίνει κλικ στο αντικείμενο που τον εκτελεί;

- A**
- ```
if (Greenfoot.mouseClicked(thisObject)) {  
}
```
- B**
- ```
if (mouseClicked(thisObject)) {  
}
```
- Γ**
- ```
if (Greenfoot.mouse(thisObject)) {  
}
```
- Δ**
- ```
if (Greenfoot.mouseClicked(this)) {  
}
```

## 8. Σκορ

Το επόμενο βήμα είναι να μετρήσουμε σε πόσα αστεράκια έκανε κλικ ο χρήστης. Έχουμε ήδη δει πώς υλοποιούμε το σκορ (Κεφάλαιο 6) αλλά σε αυτό το παιχνίδι χρειάζεται να γίνει αρκετά διαφορετικά ώστε ο χρήστης να βλέπει το σκορ στην οθόνη του.

Αν το αστέρι εξαφανίζεται την πρώτη φορά που ο χρήστης το πατάει με το ποντίκι, δεν έχει νόημα να μετρήσουμε το σκορ. Για αυτό τον λόγο το σκορ θα το μετρήσουμε αφού πρώτα υλοποιήσουμε την επέκταση 7.2. Μια υλοποίηση είναι να αντικατασταθεί η εντολή `getWorld().removeObject(this)` με τις εξής εντολές:

### Κώδικας 8.1:

```
int worldWidth=getWorld().getWidth();
int worldHeight = getWorld().getHeight();
setLocation(Greenfoot.getRandomNumber(worldWidth),
Greenfoot.getRandomNumber(worldHeight));
```

Σε μια μεταβλητή υπολογίζουμε το πλάτος του κόσμου μας (`worldWidth`) και το ύψος του (`worldHeight`). Μετά χρησιμοποιούμε την εντολή `setLocation` για να μετακινηθεί το αστέρι σε μια τυχαία θέση με μέγιστη τιμή το `worldWidth` όσον αφορά τον άξονα x, και σε μια τυχαία θέση με μέγιστη τιμή το `worldHeight` όσον αφορά τον άξονα y.

Ξέρουμε ότι το πλάτος του κόσμου (`width`) είναι 600 και το ύψος (`height`) 400 όμως δεν χρησιμοποιούμε αυτούς τους αριθμούς γιατί μπορεί να θέλουμε αργότερα να τους αλλάξουμε (π.χ. αν αποφασίσουμε ότι η εφαρμογή μας απευθύνεται σε κινητά). Με την προτεινόμενη υλοποίηση μπορούμε απλά να αλλάξουμε τις διαστάσεις στον κώδικα του `World` και δεν θα χρειαστεί να κάνουμε περαιτέρω αλλαγές.

Επειδή το σκορ είναι το ίδιο σε όλο το παιχνίδι και δεν έχει κάθε αστέρι το δικό του θα χρησιμοποιήσουμε τη λέξη-κλειδί **static** για να ορίσουμε την μεταβλητή `score`. Δηλώνοντας την μεταβλητή `score` σαν στατική η τιμή της μεταβλητής δεν συνδέεται με το αντικείμενο της κλάσης, αλλά με την ίδια την κλάση. Δεν θέλουμε η μεταβλητή `score` να είναι ορατή από άλλες κλάσεις, γιατί θέλουμε να αλλάζει μόνο όταν ένα αστέρι αλλάζει θέση. Τέλος η μεταβλητή `score` θα είναι ακέραια, δηλαδή το σκορ δεν θα έχει δεκαδικά ψηφία. Τέλος η μεταβλητή `score` αρχικά είναι 0.

Ορίζουμε λοιπόν την μεταβλητή `score` ως εξής:

### Κώδικας 8.2:

```
private static int score = 0;
```

Η μεταβλητή `score` αυξάνεται κατά 1 όταν το αστέρι αλλάζει θέση. Για να δούμε την τιμή της μεταβλητής `score` σε αυτή την φάση θα χρησιμοποιήσουμε την εντολή `System.out.println(score)`; που θα εμφανίσει ένα παράθυρο με το σκορ.

Σίγουρα θα ήταν καλύτερο το σκορ να εμφανίζεται μέσα στον κόσμο όπως γίνεται σε όλα τα παιχνίδια και όχι σε ξεχωριστό παράθυρο. Αυτό θα το κάνουμε στη συνέχεια.

### Κώδικας 8.3:

```
setLocation(Greenfoot.getRandomNumber(worldWidth),
Greenfoot.getRandomNumber(worldHeight));
score = score + 1;
System.out.println(score);
```

### Ερώτηση 8.1:

Ποιο από τα παρακάτω υπολογίζει το πλάτος του κόσμου στον οποίο υπάρχει ένα αντικείμενο.

1. `getWorld.getWidth();`
2. `getWorld().getWidth();`
3. `world.getWidth();`
4. `world.width;`

### Ερώτηση 8.2:

Τι από τα παρακάτω ισχύει για την εντολή `private static int score = 0;` (Παραπάνω από μία σωστές απαντήσεις)

1. Δημιουργεί μια μεταβλητή `score` τύπου ακεραίου.
2. Δημιουργεί μια μεταβλητή που ανήκει στο αντικείμενο (`property`)
3. Δημιουργεί μια δημόσια μεταβλητή
4. Δημιουργεί μια μεταβλητή που ανήκει στην κλάση των αντικειμένων.

### Ερώτηση 8.3:

Γιατί χρησιμοποιούμε τη λέξη κλειδί `static` στη δήλωση `private static int score = 0;`

1. Επειδή το σκορ είναι ακέραιος αριθμός.
2. Επειδή το σκορ θα φαίνεται σε στατική θέση στην οθόνη.
3. Επειδή δεν θέλουμε να αλλάζει η τιμή αυτής της μεταβλητής.
4. Επειδή το σκορ δεν αφορά ένα συγκεκριμένο αστέρι αλλά όλα τα αντικείμενα της κλάσης.

### Ερώτηση 8.4:

Τι κάνει η εντολή: `GreenfootImage image = new GreenfootImage("SCORE:", 30, new Color(200,0,0), new Color(0,0,0,0))`

1. Δημιουργεί μια εικόνα `image` που δείχνει ότι το σκορ είναι 30.
2. Δημιουργεί μια εικόνα η οποία δείχνει τη λέξη `SCORE:` με κόκκινα γράμματα
3. Δημιουργεί μια εικόνα η οποία δείχνει το σκορ (έναν αριθμό) με κόκκινα γράμματα
4. Δημιουργεί μια εικόνα η οποία δείχνει τη λέξη `SCORE:` με πράσινα γράμματα

### Ερώτηση 8.5:

Ποιο από τα παρακάτω δημιουργεί ένα κείμενο που περιλαμβάνει τη λέξη `"Score:"` μαζί με την τιμή της μεταβλητής `score`.

1. `String text = "Score: " + star.getScore();`
2. `int text = "Score: " + star.getScore();`
3. `String text = "Score: " + star.getScore();`
4. `String text = Score: + star.getScore();`

### Ερώτηση 8.6:

Ποιο θα είναι το πρόβλημα αν αλλάξουμε τη δήλωση `public static int getScore()` σε `private static int getScore()`

1. Δεν θα μπορούμε να καλέσουμε την μέθοδο από την κλάση `scoreBoard`
2. Δεν υπάρχει η λέξη κλειδί `private`
3. Δεν μπορούμε να δηλώσουμε ιδιωτικές στατικές μεθόδους.
4. Δεν θα μπορούμε να καλέσουμε την μέθοδο `getScore` από αντικείμενα της κλάσης `star`.

## 9. Γράψε στην Οθόνη

Για να εισάγουμε στον κόσμο το σκορ, θα χρειαστούμε μια νέα κλάση αντικειμένων. Θα την ονομάσουμε Scoreboard. Μέσα σε αυτή την κλάση θα χρησιμοποιήσουμε τον παρακάτω κώδικα.

### Κώδικας 9.1:

```
public class Scoreboard extends Actor
{
    private static final Color TEXT_COLOR = new
Color(200, 0, 0);
    private static final Color TRANSPARENT_COLOR = new
Color(0, 0, 0, 0);

    public Scoreboard()
    {
        updateImage();
    }
    public void act() {
        updateImage();
    }
    private void updateImage()
    {
        String text = "Score:";
        GreenfootImage image = new GreenfootImage(text,
30, TEXT_COLOR, TRANSPARENT_COLOR);
        setImage(image);
    }
}
```

Στο Greenfoot μπορεί να δημιουργηθεί η εικόνα ενός κειμένου με την εντολή GreenfootImage αρκεί αυτή να έχει ορίσματα το κείμενο (text), το μέγεθος του κειμένου (30 στη συγκεκριμένη περίπτωση) και δύο χρώματα ένα για το χρώμα των γραμμάτων και ένα για το χρώμα που θα φαίνεται στον φόντο. Τα χρώματα είναι αντικείμενα της κλάσης Color:

```
new Color(200, 0, 0);
```

Η εντολή αυτή σημαίνει πως δημιουργείται ένα χρώμα με πολύ κόκκινο χρώμα (τιμή 200 με μέγιστη τιμή 255), καθόλου πράσινο και καθόλου μπλε (μοντέλο RGB). Το χρώμα πίσω από τα γράμματα ορίζεται σε διαφανές αφού η τέταρτη τιμή που είναι το κατά πόσο το χρώμα είναι ορατό, είναι 0.

Η εντολή **setImage** βάζει αυτή την εικόνα που δημιουργήθηκε σαν εικόνα της μορφής.

Μπορείτε να βάλετε ένα αντικείμενο Scoreboard μέσα στον κόσμο και θα δείτε ότι όπου το τοποθετήσετε θα εμφανιστεί η λέξη Score. Αν θέλετε να εμφανίζεται αυτόματα θα πρέπει να εισάγετε μέσα στον κώδικα του κόσμου την εντολή:

```
addObject(new Scoreboard(),500,350);
```

Σε αυτή τη φάση το αντικείμενο Scoreboard δεν εμφανίζει το πραγματικό σκορ αλλά μόνο την λέξη Score. Χρειάζεται να πάρουμε το score από την κλάση star για να εμφανίζεται και αυτό. Μια λύση θα μπορούσε να ήταν να αλλάξουμε την δήλωση της μεταβλητής score

από `private` σε `public` και να αλλάξουμε την γραμμή `String text = "Score: ";` σε `String text = "Score: " + star.score;`.

Μια τέτοια λύση δεν προτείνεται στον αντικειμενοστραφή προγραμματισμό με την Java όπου προσπαθούμε να περιορίσουμε τα πιθανά προβλήματα. Δεδομένου ότι το σκορ αλλάζει μόνο από αντικείμενα της κλάσης `star`, δεν είναι ανάγκη να δώσουμε την δυνατότητα σε άλλα αντικείμενα να αλλάξουν αυτή την μεταβλητή, χρησιμοποιώντας `public` μεταβλητές. Μια καλύτερη λύση είναι να γυρίσουμε στην κλάση `star` και να υλοποιήσουμε μια νέα μέθοδο, την `getScore`.

#### Κώδικας 9.2:

```
public static int getScore()
{
    return score;
}
```

Μια μέθοδος μπορεί να επιστρέφει μια τιμή. Η μέθοδος `getScore` επιστρέφει τιμή ακεραίου και συγκεκριμένα την τιμή `score` (`return score`). Μάλιστα αφού η μεταβλητή `score` αφορά την κλάση `star` και όχι κάθε αντικείμενο ξεχωριστά το ίδιο δηλώνουμε και για την συνάρτηση `getScore` (δήλωση `static`). Τέλος η συνάρτηση `getScore` θέλουμε να καλείται και από άλλα αντικείμενα και γι' αυτό τον λόγο την ορίζουμε `public`.

Η τελική συνάρτηση `updateImage` της κλάσης `Scoreboard` φαίνεται παρακάτω:

#### Κώδικας 9.3:

```
private void updateImage()
{
    String text = "Score: " + star.getScore();
    GreenfootImage image = new GreenfootImage(text, 30,
    TEXT_COLOR, TRANSPARENT_COLOR);
    setImage(image);
}
```

Δεδομένου ότι η συνάρτηση `getScore` είναι δημόσια και στατική μπορούμε να την καλέσουμε με `star.getScore()`.

#### Επέκταση 9.1:



Αλλάξτε τους αριθμούς που σχηματίζουν τα χρώματα στις δηλώσεις `new Color`. Τι παρατηρείτε;

#### Επέκταση 9.2:



Βάλτε έναν τίτλο στο παιχνίδι σας. Εναλλακτικά μπορείτε να βάλετε το όνομά σας ή το ψευδώνυμό σας ώστε να φαίνεται στην οθόνη. Διαλέξτε μόνοι σας το μέγεθος και το χρώμα της γραμματοσειράς.



## 10. Εξαφάνιση

Για να αυξήσουμε το ενδιαφέρον του χρήστη, θα κάνουμε το αστέρι να εξαφανίζεται μετά από κάποιο χρόνο και σε προκαθορισμένους χρόνους θα εμφανίζονται νέα αστέρια στα οποία ο χρήστης θα πρέπει να κάνει κλικ.

Το περιβάλλον Greenfoot μας παρέχει έναν τρόπο να θέσουμε το κατά πόσο μια εικόνα θα έχει διαφάνεια ή όχι. Όταν η τιμή της διαφάνειας (transparency) είναι 255 η εικόνα φαίνεται πλήρως, όταν η διαφάνεια (transparency) πάρει την τιμή 0 τότε η εικόνα δεν φαίνεται καθόλου (πλήρης διαφάνεια).

Κατασκευάσαμε μια νέα μέθοδο `adjustTransparency`, η οποία μειώνει την ορατότητα από την μέγιστη τιμή της (255) κατά μία μονάδα σε κάθε Act. Τη νέα τιμή transparency που υπολογίζει την εφαρμόζει στην εικόνα της μορφής star χρησιμοποιώντας την εντολή: `getImage().setTransparency(transparency)` αφού πρώτα εξασφαλιστεί ότι η τιμή δεν είναι μικρότερη του μηδενός. Το Greenfoot δεν δέχεται αρνητική διαφάνεια και θα προκύψει πρόβλημα αν δεν κάνουμε αυτόν τον έλεγχο. Τέλος, για αυτή την άσκηση προγραμματίζουμε το αστεράκι να εξαφανίζεται όταν πατιέται με το ποντίκι και όχι να αλλάζει θέση.

### Κώδικας 10.1:

```
private static int score = 0;
private int transparency = 255;
public void act()
{
    // Add your action code here.

    if (Greenfoot.mouseClicked(this)) {
        getWorld().removeObject(this);
        score += 1;
    }
    adjustTransparency();
}
private void adjustTransparency() {
    getImage().setTransparency(transparency);
    if (transparency <= 0)
    {
        transparency = 0;
    }
    else
    {transparency = transparency - 1;
    }
}
public static int getScore()
{
    return score;
}
```

Δοκιμάστε να προσθέσετε περισσότερα αστέρια με την εντολή `addObject` και δείτε τι συμβαίνει. Όλα τα αστέρια εμφανίζονται και εξαφανίζονται μαζί. Καλό θα ήταν να εμφανίζονται αστέρια σε συγκεκριμένα χρονικά διαστήματα (π.χ. κάθε 50 κύκλους Act). Εφόσον η εντολή `addObject` ανήκει στην κλάση `World` μπορούμε να εισάγουμε αυτόν τον

κώδικα στην υποκλάση `space` του κόσμου `World`. Θα ορίσουμε μάλιστα και μια συνάρτηση `Act()` μέσα στην υποκλάση `space` για να εκτελείται κάθε φορά που γίνεται `Run`.

#### Κώδικας 10.2:

```
private int numOfFrames = 50;
public void act()
{
    int width = getWidth();
    int height = getHeight();
    numOfFrames--;
    if (numOfFrames == 0)
        addObject(new star(),
Greenfoot.getRandomNumber(width),
Greenfoot.getRandomNumber(height));
    numOfFrames = 50;
}
```

#### Επέκταση 10.1:



Εισάγετε έναν νέο τύπο αντικειμένων και δώστε την ίδια λειτουργία με τα αντικείμενα `star`. Προσπαθήστε να κάνετε το παιχνίδι πιο εύκολο ή πιο δύσκολο αλλάζοντας το πόσο συχνά εμφανίζονται τα αντικείμενα αυτά στην οθόνη. (Δεν χρειάζεται να αλλάζει το σκορ κάθε φορά που ο χρήστης πατάει το νέο αντικείμενο που θα εισάγετε).

#### Επέκταση 10.2:



Μπορείτε να προγραμματίσετε το παιχνίδι ώστε όσο πιο γρήγορα πατήσει ο παίκτης το αστέρι, τόσο πιο πολύ να αυξάνει το σκορ; Σκεφτείτε ότι η τιμή της μεταβλητής `transparency` μειώνεται σε κάθε `act()`.

## 11. Αντίστροφη μέτρηση

Στα περισσότερα παιχνίδια, όταν ο χρήστης δεν κάνει τίποτε τότε μετά από κάποιο χρόνο χάνει. Αυτό δεν συμβαίνει ακόμη στο δικό μας παιχνίδι. Για να υλοποιήσουμε αυτή τη λειτουργία θα εισάγουμε αντίστροφη μέτρηση στο παιχνίδι μας.

Θα εισάγουμε μια νέα υποκλάση `Timer` η οποία θα μετράει κάποια δευτερόλεπτα αντίστροφα και όταν αυτά τα δευτερόλεπτα περάσουν, το παιχνίδι θα τελειώσει. Η υποκλάση αυτή θα δείχνει τα δευτερόλεπτα στην οθόνη οπότε ο κώδικας που θα υλοποιήσουμε θα μοιάζει με αυτόν της κλάσης `Scoreboard`.

### Κώδικας 11.1:

```
private static final Color TEXT_COLOR = new Color(200, 0, 0);
private static final Color TRANSPARENT_COLOR = new Color(0, 0, 0, 0);
```

```
public Timer()
{
    updateImage();
}
public void act() {
    updateImage();
}
private void updateImage()
{
    String text = "Seconds left: ";
    GreenfootImage image = new GreenfootImage(text, 30,
TEXT_COLOR, TRANSPARENT_COLOR);
    setImage(image);
}
```

Για να υπολογίσουμε τον χρόνο του ενός δευτερολέπτου υπάρχει η μέθοδος `currentTimeMillis` που ανήκει στην κλάση `System`.

```
long System.currentTimeMillis()
```

Η μέθοδος αυτή επιστρέφει αριθμούς τύπου `long` δηλαδή πολύ μεγάλους ακεραίους, συγκεκριμένα επιστρέφει τα χιλιοστά του δευτερολέπτου που έχουν περάσει από τα μεσάνυχτα της 1/1/1970. Αρχικά σε μια μεταβλητή θα αποθηκευτεί ο τρέχων χρόνος που θα είναι ο αρχικός χρόνος.

```
timeStarted = System.currentTimeMillis();
```

Σε κάθε `act` θα ελέγχουμε αν η διαφορά μεταξύ του τρέχοντος χρόνου και του αρχικού χρόνου είναι 1000 milliseconds. Όταν η διαφορά θα είναι μεγαλύτερη από 1000 milliseconds αυτό θα σημαίνει ότι έχει περάσει ένα δευτερόλεπτο. Θα πρέπει τότε να ορίσουμε τον αρχικό χρόνο στον τρέχων χρόνο για να μπορέσουμε να μετρήσουμε ένα νέο δευτερόλεπτο.

```
if (System.currentTimeMillis() - timeStarted > 1000) {
    timeStarted = System.currentTimeMillis();
}
```

Για να μετρήσουμε δέκα δευτερόλεπτα θα χρειαστούμε μια δεύτερη μεταβλητή με όνομα `seconds`. Την αρχικοποιούμε στο 10 και κάθε φορά που περνάει ένα δευτερόλεπτο τη μειώνουμε κατά 1. Ο χρήστης θα πρέπει να ενημερωθεί για την τιμή του μετρητή, για αυτό το λόγο θα πρέπει να αλλάξει η εντολή `String text = "Seconds left: "` σε `String text = "Seconds left: " + seconds;` ώστε να φαίνονται και τα δευτερόλεπτα που έχουν περάσει.

### Κώδικας 11.2:

```
private static final Color TEXT_COLOR = new Color(200,
0, 0);
private static final Color TRANSPARENT_COLOR = new
Color(0, 0, 0, 0);
private long timeStarted;
private int seconds = 10;
public Timer()
{
    updateImage();
    timeStarted = System.currentTimeMillis();
}
public void act() {
    updateImage();
    if (System.currentTimeMillis()-timeStarted>1000) {
        seconds--;
        timeStarted = System.currentTimeMillis();
    }
}
private void updateImage()
{
    String text = "Seconds left: " + seconds;
    GreenfootImage image = new GreenfootImage(text,
30, TEXT_COLOR, TRANSPARENT_COLOR);
    setImage(image);
}
```

Παρατηρούμε ότι το παιχνίδι δεν τελειώνει ποτέ, καθώς ο μετρητής συνεχίζει να μετράει αρνητικούς αριθμούς. Χρειάζεται να προσθέσουμε έναν τελευταίο έλεγχο στο παιχνίδι μας. Αν η μεταβλητή seconds γίνει 0 τότε θα πρέπει το παιχνίδι να σταματήσει. Το παιχνίδι σταματάει με την εντολή Greenfoot.stop().

### Κώδικας 11.3:

```
public void act() {
    updateImage();
    if (System.currentTimeMillis()-timeStarted>1000) {
        seconds--;
        if (seconds == 0)
        {
            Greenfoot.stop();
        }
        timeStarted = System.currentTimeMillis();
    }
}
```

### Επέκταση 11.1:



Δώστε περισσότερο χρόνο στον παίκτη. Τι θα αλλάξετε ώστε ο παίκτης να παίζει για 20 δευτερόλεπτα.