Ευχαριστίες

Η παρούσα διπλωματική εργασία ολοκληρώνει τις σπουδές μου στο Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων (ΤΜΗΥΤΔ) του Πανεπιστημίου Θεσσαλίας. Θα ήθελα να ευχαριστήσω την επιβλέπουσα καθηγήτριά μου, κυρία *Ασπασία Δασκαλοπούλου*, για το ενδιαφέρον θέμα που μου ανέθεσε, την καθοδήγηση και την υπομονή που επέδειξε καθ' όλη τη διάρκεια εκπόνησης της εργασίας.

Βόλος, Ιούνιος 2012

Περιεχόμενα

1

ΕΙΣΑΓΩΓΗ

1.1 Γενικά

Το ρομπότ είναι μια μηχανική συσκευή η οποία μπορεί να υποκαθιστά τον άνθρωπο σε διάφορες εργασίες. Ένα ρομπότ μπορεί να δράσει κάτω από τον απ' ευθείας έλεγχο ενός ανθρώπου ή αυτόνομα κάτω από τον έλεγχο ενός προ – προγραμματισμένου υπολογιστή.

Τα ρομπότ μπορούν να χρησιμοποιηθούν ώστε να κάνουν εργασίες οι οποίες είτε είναι δύσκολες ή επικίνδυνες για να γίνουν απ' ευθείας από έναν άνθρωπο. Σε άλλες περιπτώσεις, χρησιμοποιούνται για να εκτελέσουν εργασίες ταχύτερα ή φθηνότερα απ' ότι ο άνθρωπος. Έτσι, μπορούν να χρησιμοποιηθούν στην αυτόματη παραγωγή μεγάλων ποσοτήτων κάποιου προϊόντος και με χαμηλότερο κόστος (για παράδειγμα, στις αλυσίδες παραγωγής).

Τα ανθρωποειδή ρομπότ θα έχουν γίνει μέρος της κοινωνίας μας μέχρι το 2050, υποστηρίζει επιστήμονας εξειδικευμένος στα ρομπότ, ο δόκτορας Frank Pollick. Η εποχή των ανθρωποειδών ξημερώνει. Τα

ρομπότ θα μπορούν να ασχολούνται με οικιακά, με εργασίες γραφείου, θα βοηθούν στα νοσοκομεία ή ακόμα και στον αθλητισμό.

Σήμερα πάνω από 950.000 ρομπότ εργάζονται στις βιομηχανίες όλου του κόσμου, ακόμα και στα χειρουργεία νοσοκομείων. Αν όμως μεταφερθούμε από τα εργοστάσια στα ερευνητικά εργαστήρια, το τοπίο αλλάζει εντελώς: ρομπότ παίζουν ντραμς, ανθρωποειδή χορεύουν, ανδροειδή παίζουν μπάλα, πιθηκορομπότ κάνουν σάλτους και τεχνητά κεφάλια γελούν σαν μωρά μπροστά σε μια κούκλα. Ένας ολόκληρος ζωολογικός κήπος από τεχνητά πλάσματα, που μπορούν να κινούνται, να εκφράζουν συναισθήματα και μερικές φορές να τρώνε και να χωνεύουν σαν εμάς, ή σχεδόν... Οι σχεδιαστές τους αντιγράφουν τον άνθρωπο (ανθρωποειδή ρομπότ) ή εμπνέονται από τα ζώα (ζωοειδή ρομπότ), χρησιμοποιούν ακόμα και ζωντανά όντα για να φτιάξουν βιορομπότ (υβρίδια από μηχανικά και βιολογικά μέρη). Στόχος πολλών ερευνητών δεν είναι απλά να κατασκευάσουν τέλειους μηχανικούς σκλάβους. Φιλοδοξούν κάτι περισσότερο: να χαρίσουν στον άνθρωπο τεχνητούς φίλους αρκετά αυτόνομους και ευφυείς ώστε να είναι άξιοι σύντροφοί του.

1.2 Περιγραφή Προβλήματος

Η κοινωνική νοημοσύνη είναι κλάδος της θεωρίας της νοημοσύνης. Η ανάπτυξη αυτού του κλάδου μπορούσε ως τώρα να γίνει από την πλευρά της κοινωνιολογίας και της ψυχολογίας. Αλλά σήμερα όπως δείχνουν τα πράγματα θα υπάρξει τεράστια ανάπτυξη και από την πλευρά της πληροφορικής μέσα από τον κλάδο της τεχνητής νοημοσύνης. Στην πραγματικότητα όμως υπερβαίνει τα όρια της τεχνητής νοημοσύνης, γιατί συνδυάζει αλγοριθμικές τεχνικές με τις γνώσεις και τις θεωρίες από την θεωρία της συμπεριφοράς καθώς και με στοιχεία από την αλληλεπίδραση κοινωνικών ομάδων και τα εφαρμόζει σε τεχνολογικά συστήματα.

Όπως είναι φυσικό, η επιστήμη της ψυχολογίας παίζει καταλυτικό ρόλο στην ανάπτυξη των συναισθηματικών ρομπότ – καθώς δημιουργεί τα μοντέλα ανθρώπινης συμπεριφοράς, τα οποία χρησιμοποιούν οι προγραμματιστές για να δημιουργήσουν μηχανές που θα καταφέρνουν να αποκωδικοποιούν τα συναισθήματα του χρήστη τους. Ωστόσο, θα είναι και από τους πρώτους κλάδους που θα ωφεληθούν από τη χρήση τέτοιων ψηφιακών συστημάτων.

«Μέχρι σήμερα, στις περισσότερες περιπτώσεις οι ψυχολόγοι βασίζουν τα συμπεράσματά τους στα ερωτηματολόγια, τα οποία συμπληρώνουν οι συμμετέχοντες μετά το τέλος των πειραμάτων», λέει ο ερευνητής του ΕΜΠ κ. Κ. Καρπούζης, «ενώ τα "συναισθηματικά ευφυή" μηχανήματα θα τους δίνουν τη δυνατότητα να ελέγξουν απευθείας τις αντιδράσεις των εθελοντών».

Ο στόχος σήμερα -η κατασκευή νοημόνων μηχανών- είναι ίδιος μ' εκείνον των πρώτων ερευνών για τη δημιουργία τεχνητής νοημοσύνης, όμως η μέθοδός του βασίζεται σε δύο εντελώς διαφορετικές αρχές. Πρώτον, δεν μπορεί να υπάρξει νοημοσύνη έξω από ένα σώμα και, δεύτερον, για να μπορέσουμε να την αναπαράγουμε πρέπει οι ίδιες οι μηχανές να μάθουν ν' αντιδρούν από μόνες τους σε κάθε νέα κατάσταση που αντιμετωπίζουν -όπως ακριβώς κάνει ένα μωρό όταν αρχίζει να εξερευνά τον κόσμο. Καμιά μορφή νοημοσύνης δεν είναι άυλη και ανεξάρτητη από το σώμα που την παράγει και επομένως θα ήταν μάταιο να προσπαθούμε να την εισαγάγουμε «άνωθεν» στις μηχανές. Αυτή λοιπόν η νέα προσέγγιση της νοημοσύνης των μηχανών, «από κάτω προς τα πάνω», αντιτίθεται στην αφηρημένη προσέγγιση «από πάνω προς τα κάτω» της τεχνητής νοημοσύνης.

1.3 Η Γενικότερη Εφαρμογή

Τα συναισθήματα παίζουν καίριο λόγο και στη διαδικασία λήψης αποφάσεων. Ασθενείς που πάσχουν από ένα συγκεκριμένο είδος

εγκεφαλικής βλάβης χάνουν την ικανότητα να νοιώθουν το παραμικρό αίσθημα. Η νοητική τους ικανότητα μένει ανέπαφη, όμως δεν μπορούν να εκφράσουν κανένα συναίσθημα. Ο νευρολόγος Δρ.Antonio Damasio, της Ιατρικής Σχολής του Πανεπιστημίου της Αιόβα που έχει μελετήσει αυτή την κατηγορία ασθενών, καταλήγει στο συμπέρασμα ότι είναι σαν «να γνωρίζουν αλλά να μην αισθάνονται». Χωρίς συναισθήματα να τους καθοδηγούν, σταθμίζουν αέναα τα υπέρ και τα κατά της κάθε επιλογής και το αποτέλεσμα είναι μια αναποφασιστικότητα που τους παραλύει. Οι υπόλοιποι λαμβάνουμε πολλές φορές τις αποφάσεις μας με βάση το «προαίσθημα» ή το «ένστικτό» μας. Οι ασθενείς που έχουν υποστεί βλάβες στο σύστημα επικοινωνίας του λογικού και του συναισθηματικού τμήματος του εγκεφάλου, δεν διαθέτουν αυτή την ικανότητα.

Όταν βγαίνουμε στην αγορά για παράδειγμα, υποσυνείδητα κάνουμε χιλιάδες αξιολογικές κρίσεις για κάθε τι που βλέπουμε. Σκεφτόμαστε, ας πούμε, ότι κάτι είναι πολύ ακριβό, πολύ φτηνό, πολύ φανταχτερό, πολύ χαζό, ή ακριβώς αυτό που θέλουμε. Για τους ασθενείς που πάσχουν από τη συγκεκριμένη εγκεφαλική βλάβη, τα ψώνια μπορεί ν’ αποδειχτούν εφιάλτης, αφού γι’ αυτούς όλα φαίνονται να έχουν την ίδια αξία. Καθώς τα ρομπότ θα γίνονται ευφυή και θα μπορούν να κάνουν ανεξάρτητα τις επιλογές τους, ίσως αντιμετωπίσουν το ίδιο πρόβλημα : την παράλυση που προκαλείται από την αναποφασιστικότητά τους (αυτό μας θυμίζει την παραβολή με το γάιδαρο που στέκεται μπροστά σε δύο μπάλες σανού και στο τέλος πεθαίνει από ασιτία, επειδή δεν μπορεί ν’ αποφασίσει ποια να φάει). Σχολιάζοντας την έλλειψη συναισθημάτων των ρομπότ, ο Δρ. Rosalind Picard, του Εργαστηρίου Media του MIT, λέει : «Δεν μπορούν να ιεραρχήσουν τι είναι πιο σημαντικό. Αυτή είναι μια από τις μεγαλύτερες αδυναμίες τους. Απλούστατα, οι υπολογιστές δεν σκαμπάζουν». Με άλλα λόγια, τα ρομπότ του μέλλοντος ίσως χρειαστούν τα συναισθήματα για να μπορούν να θέτουν στόχους και για να δίνουν νόημα και τάξη στη «ζωή» τους. Διαφορετικά, θα παραλύουν από τις άπειρες εναλλακτικές επιλογές τους.

«Τα συναισθήματα παίζουν έναν πολύ σημαντικό ρόλο στην καθοδήγηση και την αναγνώριση του τί είναι σημαντικό», σχολιάζει η καθηγήτρια ρομποτικής Cynthia Breazeal, από το Ινστιτούτο Τεχνολογίας της Μασαχουσέτης. «Επιτρέπουν στο ρομπότ να λαμβάνει σωστότερες αποφάσεις, να μαθαίνει και να αντιδρά πιο αποτελεσματικά».

Προς το παρόν, τα ρομπότ που είναι διαθέσιμα στην αγορά είναι ικανά να πραγματοποιήσουν μονάχα μία εργασία. Όμως μία σειρά από ειδικούς στη ρομποτική οι οποίοι μίλησαν στο συνέδριο της *American Association for the Advancement of Science* στο Σαν Φραντζέσκο είπαν ότι μέσα σε 10 χρόνια θα είναι διαθέσιμα ρομπότ τα οποία θα είναι ικανά για πολλαπλές λειτουργίες ενώ θα μπορούν επίσης να παρέχουν συντροφιά στον ιδιοκτήτη τους.

Οι επιστήμονες ισχυρίζονται ότι κάτι τέτοιο είναι ήδη εφικτό. Αρκετές επιστημονικές ομάδες ανά τον κόσμο σήμερα κατασκευάζουν ρομπότ τα οποία διαθέτουν τα βασικά συναισθήματα. Εάν ένα ρομπότ νιώθει ευτυχισμένο αφού έχει καθαρίσει πολύ καλά μία βρώμικη μοκέτα τότε θα θελήσει να κάνει το ίδιο και σε μία άλλη βρώμικη επιφάνεια. Παρομοίως εάν το ρομπότ νιώσει ένοχο ή λυπημένο εφόσον έχει αποτύχει σε μία αποστολή, την επόμενη φορά θα προσπαθήσει πιο πολύ.

1.4 Οργάνωση Κειμένου

<u>Κεφάλαιο 2 (υπόβαθρο)</u>

- Γίνεται περίληψη των προγραμμάτων MyPet που σχεδιάστηκαν.
- Αναφορά άλλων περιπτώσεων-εφαρμογών παρόμοιων με το MyPet.

Κεφάλαιο 3 (σχεδιασμός / υλοποίηση)

- Class Diagrams.
- Sequence/Activity Diagrams (UML).
- Προσδιορισμός γλώσσας -ιδιαίτερα προγραμματιστικά σημεία που χρησιμοποιήθηκαν στον προγραμματισμό-.
- Επιλεκτικά σημεία κώδικα για κυριότερες λειτουργίες.
- Εκτελέσεις με επιλεκτικά print screens, συνοδευόμενα από οδηγίες.

Κεφάλαιο 4 (συμπεράσματα / αποτίμηση / μελλοντική δουλειά)

- Περιγραφή του τι έκανα στην εργασία.
- Τι προσθήκες σε χαρακτηριστικά θα μπορούσαν να γίνουν στο μέλλον.

Κεφάλαιο 5 (βιβλιογραφία)

2

ΥΠΟΒΑΘΡΟ

2.1 MyPet ΠΕΡΙΛΗΨΗ

2.1.1 MyPet v1.0

Στη πρώτη βασική έκδοση του προγράμματος MyPet υλοποιείται το κατοικίδιο με το οποίο αλληλεπιδράμε μέσα από την κονσόλα. Έχει χρόνο ζωής 1 λεπτού. Το κατοικίδιο εκτός από τις ενέργειες *να γαυγίζει, να δαγκώνει* και *να κουνάει την ουρά του* έχει χαρακτηριστικά όπως *πείνα, υγεία* και *ευτυχία* τα οποία επηρεάζονται με το πέρασμα του χρόνου και ανάλογα με τις αλληλεπιδράσεις μαζί μας.

2.1.2 MyPet v2.0

Στη δεύτερη και ολοκληρωμένη έκδοση του προγράμματος MyPet υλοποιείται το κατοικίδιο το οποίο όμως τώρα αλληλεπιδρά, αντί για εμάς με μια άλλη οντότητα-πράκτορα Avatar. Το MyPet έχει τα ίδια χαρακτηριστικά με πριν, ενώ ο Avatar έχει σαν χαρακτηριστικά την *ευτυχία* και το *πόσο απασχολημένος* είναι. Το MyPet και ο Avatar ανταλλάσουν ενδείξεις, οι οποίες οδηγούν σε ανάλογες ενέργειες και από τις 2 πλευρές.

2.2 Παρόμοιες εφαρμογές με MyPet

2.2.1 Farmville

Το Farmville (εικόνα 1) πρόκειται για μια ηλεκτρονική φάρμα. Είναι ένα παιχνίδι προσομοίωσης πραγματικού χρόνου που μπορείς να έχεις τη δική σου φάρμα με τα ζώα, τις καλλιέργειες και τα δέντρα που επιθυμείς. Όσο αρχίζεις και ανεβαίνεις επίπεδα όμως, έχεις όλο και περισσότερες δυνατότητες. Σταδιακά, μπορείς να χτίσεις μία αποθήκη ή το κοτέτσι σου, να αγοράσεις τα πιο σπάνια είδη ζώων και δέντρων, να πάρεις τρακτέρ για τη συγκομιδή της σοδειάς σου ή να επεκτείνεις τη φάρμα σου! Σε αυτό μάλιστα στηρίζονται πολύ και οι παραγωγοί του παιχνιδιού, διότι για να επεκτείνεις τη φάρμα πρέπει να έχεις αρκετούς «γείτονες» στο παιχνίδι.

Αν μια καλλιέργεια δεν συγκομίζεται στο διπλάσιο του χρόνου ανάπτυξης, μαραίνεται και πρέπει ο χρήστης να καλλιεργήσει και να φυτέψει πάλι τη γη. Για παράδειγμα, οι κολοκύθες θα μεγαλώνουν σε 8 ώρες. Αν δεν συγκομίζονται εντός 16 ωρών από τη φύτευση, θα αρχίσουν να μαραίνονται και να πεθαίνουν. Ο παίκτης έτσι δεν θα είναι σε θέση να κάνει την συγκομιδή.

2.2.2 Ταμαγκότσι

Το Tamagotchi (εικόνα 2) είναι ένα μικρό φορητό ηλεκτρονικό ζωάκι, σχήματος αυγού, και πωλείται από την εταιρεία Bandai. Αποτελεί ιδέα τού Ιάπωνα Aki Maita, υπάλληλου της εταιρείας Bandai το 1996. Διαθέτει τρία πλήκτρα (A, B και C) για την επιλογή των ακόλουθων βασικών λειτουργιών:

- Τάισμα τού χαρακτήρα.
- Παίξιμο παιχνιδιού με το χαρακτήρα.
- Καθάρισμα τού χαρακτήρα.
- Έλεγχος ηλικίας, αταξίας, πείνας και ευτυχίας του χαρακτήρα.
- Επικοινωνία με άλλα Ταμαγκότσι μέσω υπέρυθρων.

Η έννοια της λέξης προέρχεται από την ιαπωνική λέξη (tamago) που σημαίνει *αυγό* και την αγγλική λέξη *watch* (γουότς) που σημαίνει *ρολόι*.

2.2.3 Autom

Έχει γαλάζια μάτια, μελαγχολικό βλέμμα, ύψος μόλις 38 εκατοστά και είναι ο ηλεκτρονικός «σύντροφος» στη μάχη ενάντια στα περιττά κιλά.

Το ρομπότ που ονομάζεται Autom (εικόνα 3), μιλά άπταιστα αγγλικά και κινέζικα και μπορεί να καταγράψει δεκάδες χιλιάδες διαφορετικά στοιχεία, να προτείνει προγράμματα διατροφής και να παρέχει ψυχολογική υποστήριξη στον ιδιοκτήτη του.

Το μόνο που χρειάζεται να κάνει ο χρήστης είναι να θέσει το ρομπότ σε λειτουργία και από εκεί και πέρα ο Autom δίνει όλες τις απαραίτητες οδηγίες για τη συνέχεια. Το ρομπότ αρχίζει να υποβάλλει ερωτήσεις για το βάρος, τις διατροφικές συνήθειες, την αθλητική δραστηριότητα και τους στόχους που θέτει ο χρήστης και καταγράφει

τα στοιχεία στη μνήμη του μαζί με τυχόν αδυναμίες και ιδιαιτερότητες του καινούριου του «αφεντικού».

Σταδιακά και καθημερινά ο Autom συγκεντρώνει νέα στοιχεία που του «φανερώνουν» τα αδύνατα και τα δυνατά στοιχεία του ανθρώπου που έχει απέναντί του και έτσι μπορεί να υποβάλει τα σχετικά ερωτήματα και να δίνει τις κατάλληλες συμβουλές. Ακόμη, το ρομπότ μπορεί να δημιουργήσει γραφικές απεικονίσεις για την πρόοδο της δίαιτας.

Ο Autom, που κατασκευάζεται από την εταιρεία Automata στο Χονγκ Κονγκ, ανήκει στην κατηγορία των «κοινωνικών» ρομπότ, τα οποία προσαρμόζουν τη συμπεριφορά τους ανάλογα με τον άνθρωπο που έχουν απέναντί τους. Πριν αποφασιστεί η εμπορική εκμετάλλευση του ρομπότ, πραγματοποιήθηκε μια έρευνα στην περιοχή της Βοστόνης, με τη συμμετοχή 45 ατόμων που ήθελαν να χάσουν βάρος. Δεκαπέντε άτομα είχαν τη βοήθεια του Autom στην προσπάθειά τους, που κράτησε 51 ημέρες, η δεύτερη ομάδα βασίστηκε σε έναν υπολογιστή που διέθετε ακριβώς το ίδιο λογισμικό με το ρομπότ και η τρίτη έπρεπε να αρκεστεί σε οδηγίες τυπωμένες σε χαρτί.

Ολοι οι συμμετέχοντες έπρεπε να ακολουθήσουν συγκεκριμένο πρόγραμμα διατροφής και άσκησης. Τα αποτελέσματα έδειξαν ότι όσοι «συνεργάστηκαν» με τον Autom άντεξαν 40% περισσότερο από τους χρήστες του υπολογιστή και σχεδόν το διπλάσιο από όσους είχαν μόνο έντυπο υλικό.

Ο εφευρέτης του ρομπότ συνιστά στους χρήστες να του δώσουν κάποιο όνομα, να του φορέσουν ένα καπέλο ή ένα φουλάρι έτσι ώστε να δημιουργήσουν την ψευδαίσθηση ότι πρόκειται για ένα ζωντανό πλάσμα, ώστε να μην αρχίσουν να αδιαφορούν για τις νουθεσίες του Autom μόλις ξεθωριάσει η γοητεία του καινούργιου αποκτήματος.

Στόχος του τα επόμενα χρόνια είναι να βελτιώσει την εμφάνιση του ρομπότ, αλλάζοντας το πρόσωπο και το κορμί του, αλλά επίσης κάνοντας αναβαθμίσεις στον τρόπο συμπεριφοράς του, έτσι ώστε η προσωπικότητά του να είναι ακόμη πιο εύκολα αποδεκτή και αρεστή.

Το επόμενο βήμα είναι η εξέλιξη του Autom σε μεγαλύτερο βαθμό, έτσι ώστε να μπορεί να ανταποκριθεί στις ανάγκες ανθρώπων που αντιμετωπίζουν προβλήματα υγείας, όπως τοξικομανείς που βρίσκονται στο στάδιο απεξάρτησης, αλκοολικοί αλλά και διαβητικοί.

2.2.4 EVER-1

Προς την κατεύθυνση αυτή σημαντική ήταν η συμβολή της Κορέας, η οποία παρουσίασε το δικό της ανδροειδές ικανό να αναπαράγει συναισθήματα με το πρόσωπό του. Το όνομα αυτού Ever-1 (εικόνα 4), μία σύντμηση των λέξεων Εύα (Eve) και ρομπότ (robot).

Η Ever-1 δημιουργήθηκε από το Κορεάτικο Ινστιτούτο Βιομηχανικής Τεχνολογίας (KITECH) με τα χαρακτηριστικά μιας γυναίκας 20 χρονών, ύψους 1,60μ και βάρους 50 κιλών. Δεν είναι αρκετά ευκίνητη μιας και μπορεί να κουνήσει μόνο το πάνω μέρος του σώματός της. Μπορεί να απεικονίσει στο τεχνητό πρόσωπο συναισθήματα όπως θυμό, ευτυχία, λύπη και ικανοποίηση. Το «δέρμα» της είναι φτιαγμένο από τζελ σιλικόνης με αίσθηση παρόμοια ενός ανθρώπου. Έχει την ικανότητα να κατανοεί γύρω στις 300 λέξεις και να καταλαβαίνει αν κάποιος βρίσκεται κοντά της.

2.2.5 Kansei

Ακολουθώντας πιστά τον κανόνα, ο οποίος τους θέλει να πρωτοπορούν σε κάθε τεχνολογική εξέλιξη, οι Ιάπωνες κάνουν ακόμη ένα βήμα προς το μέλλον, παρουσιάζοντας τη νέα τους δημιουργία. Ο Κανσέι (εικόνα 5) είναι ένα ρομπότ το οποίο έχει τη δυνατότητα να παίρνει 36 διαφορετικές ανθρώπινες εκφράσεις, αναλόγως του τι ακούνε τα «αυτιά» του (δηλαδή οι αισθητήρες), ενώ στο μέλλον θα μπορεί ακόμη και να μιλά.


Bomb

3

ΣΧΕΔΙΑΣΜΟΣ / ΥΛΟΠΟΙΗΣΗ

3.1 MyPet

3.1.1 MyPet v1.0

CHARACTERISTICS of MyPet

Το κατοικίδιο έχει 3 χαρακτηριστικά:

- Hunger (Πείνα)
- Health (Υγεία)
- Happiness (Ευτυχία)

που παίρνουν τιμές από 0 μέχρι 5.

Στην αρχή τα 3 χαρακτηριστικά παίρνουν τυχαίες τιμές.

ACTIONS of MyPet

Το κατοικίδιο μπορεί να πραγματοποιήσει τις παρακάτω ενέργειες:

- Bark (γαύγισμα)

Unconditionally

Γαυγίζει τυχαία, χωρίς κάποιο λόγο.

Conditionally

Γαυγίζει υπό συνθήκη. Δηλαδή Bark (X), όπου X είναι η αιτία. Δηλαδή πείνα, αρρώστια, δυστυχία.

Αρχικά να γαβγίζει όταν

[Hunger>3, Health<3, Happiness<3]

αλλά ο χρήστης μπορεί να παραμετροποιήσει τις αιτίες που θα προκαλούν το γάβγισμα του MyPet ανάλογα με την ένταση τους.

- Tail (κούνημα ουράς)

Unconditionally

Κουνάει την ουρά του τυχαία, χωρίς κάποιο λόγο.

Conditionally

Κουνάει την ουρά του υπό συνθήκη. Δηλαδή Tail (X), όπου X είναι η αιτία. Δηλαδή η ευτυχία. Πιθανή παραμετροποίηση του κουνήματος ουράς ανάλογα με ένταση της ευτυχίας.

Κουνάει την ουρά του όταν [Happiness>3]

- Bite (δάγκωμα)

Conditionally

Δαγκώνει υπό συνθήκη. Δηλαδή Bite (X), όπου X είναι η αιτία. Δηλαδή πείνα, δυστυχία.

Δαγκώνει όταν [Happiness=0, Hunger=5]

- Dies (θάνατος)

Πεθαίνει όταν [Health=0]

TIMER of MyPet

Το πρόγραμμα έχει διάρκεια ζωής 1 λεπτού.

Κάθε 5 δευτερόλεπτα η τιμή Hunger αυξάνεται κατά 1.

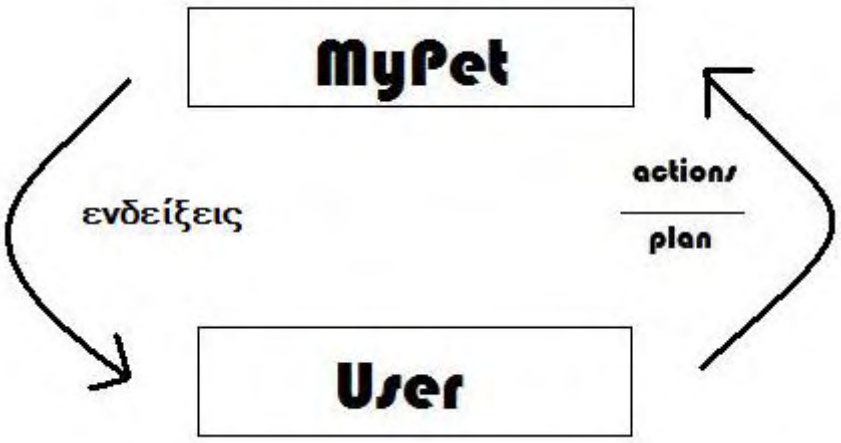
Κάθε 8 δευτερόλεπτα η τιμή Happiness μειώνεται κατά 1.

Κάθε 12 δευτερόλεπτα η τιμή Health μειώνεται κατά 1.

TIMER for ConsoleInput

Ο χρήστης κάθε 15 δευτερόλεπτα πρέπει να προσφέρει φαγητό στο MyPet με είσοδο απο την Κονσόλα. Το φαγητό είναι είτε

- Κρέας (μειώνει hunger -2, αυξάνει happiness +1, μειώνει health -1)
- Σαλάτα (μειώνει hunger -0,5, αυξάνει happiness +0.5, αυξάνει health +1)
- Γλυκό (μειώνει hunger -1, αυξάνει happiness +1.5, μειώνει health -1)


MyPet
ενδείξεις
actions
plan
User

3.1.2 MyPet v2.0

CHARACTERISTICS of MyPet

Το κατοικίδιο θα έχει 3 χαρακτηριστικά

- **Hunger** (Πείνα)
- **Health** (Υγεία)
- **Happiness** (Ευτυχία)

που θα παίρνουν τιμές από 0 μέχρι 5.

Στην αρχή τα 3 χαρακτηριστικά παίρνουν τυχαίες τιμές.

ACTIONS of MyPet

Το κατοικίδιο μπορεί να πραγματοποιήσει τις παρακάτω ενέργειες:

- **Bark (γαύγισμα)**

Unconditionally

Γαυγίζει τυχαία, χωρίς κάποιο λόγο.

Conditionally

Γαυγίζει υπό συνθήκη. Δηλαδή Bark (X), όπου X είναι η αιτία. Δηλαδή πείνα, αρρώστια, δυστυχία. *Πιθανή παραμετροποίηση ανάλογα με ένταση αιτίας.*

Γαβγίζει όταν [Hunger>3, Health<3, Happiness<3]

- **Tail (κούνημα ουράς)**

Unconditionally

Κουνάει την ουρά του τυχαία, χωρίς κάποιο λόγο.

Conditionally

Κουνάει την ουρά του υπό συνθήκη. Δηλαδή Tail (X), όπου X είναι η αιτία. Δηλαδή ευτυχία. *Πιθανή παραμετροποίηση ανάλογα με ένταση αιτίας.*

Κουνάει την ουρά του όταν [Happiness>3]

- **Bite (δάγκωμα)**

Conditionally

Δαγκώνει υπό συνθήκη. Δηλαδή Bite (X), όπου X είναι η αιτία. Δηλαδή πείνα, δυστυχία.

Δαγκώνει όταν [Happiness=0, Hunger=5]

- **Dies (θάνατος)**

Πεθαίνει όταν [Health=0] το πρόγραμμα τερματίζει

<u>TIMER of MyPet</u>

Το πρόγραμμα θα έχει διάρκεια ζωής 1 λεπτού.

Κάθε 5 δευτερόλεπτα η τιμή Hunger αυξάνεται κατα 1.

Κάθε 8 δευτερόλεπτα η τιμή Happiness μειώνεται κατα 1.

Κάθε 12 δευτερόλεπτα η τιμή Health μειώνεται κατα 1.

Ο Χρήστης (User)

Έχουμε αυτοματοποίηση του χρήστη (avatar)

Τώρα δεν είμαστε εμείς ο χειριστής του MyPet αλλά είναι μια ξεχωριστή οντότητα του προγράμματος (User) που αλληλεπιδρά με το MyPet. Ο user δέχεται σαν είσοδο τα χαρακτηριστικά του MyPet.

Χαρακτηριστικά User

- Happiness
- Busy

παίρνουν τιμές από 0 μέχρι 5.

Στην αρχή τα 2 χαρακτηριστικά παίρνουν τυχαίες τιμές.

Τα χαρακτηριστικά αυξομειώνονται (+-0,5) με (random) τυχαίο τρόπο κατά την διάρκεια του ενός λεπτού που τρέχει το πρόγραμμα. Έστω ότι ο έλεγχος για την αυξομείωση γίνεται κάθε 10 δευτερόλεπτα.

Το Happiness αυξάνεται κατά 0,5 όταν το MyPet κουνάει την ουρά του.

Το Busy είναι ανεξάρτητο από το Mypet.

TIMER για ενέργειες User

Ο χρήστης κάθε 15 δευτερόλεπτα πρέπει να προσφέρει φαγητό στο MyPet. Το φαγητό είναι είτε:

- Κρέας (μειώνει hunger -2, αυξάνει happiness +1, μειώνει health -1)

- Σαλάτα (μειώνει hunger -0,5, αυξάνει happiness +0.5, αυξάνει health +1)
- Γλυκό (μειώνει hunger -1, αυξάνει happiness +1.5, μειώνει health -1)

Random επιλογή Φαγητού.

Ο χρήστης κάθε 20 δευτερόλεπτα πρέπει να συναναστρέφεται με το MyPet.

- Καλή μεταχείριση (αυξάνει happiness +2, αυξάνει health +0.5)
- **Κακή μεταχείριση (μειώνει happiness -2, μειώνει health -0.5)**

αν User Happiness<2

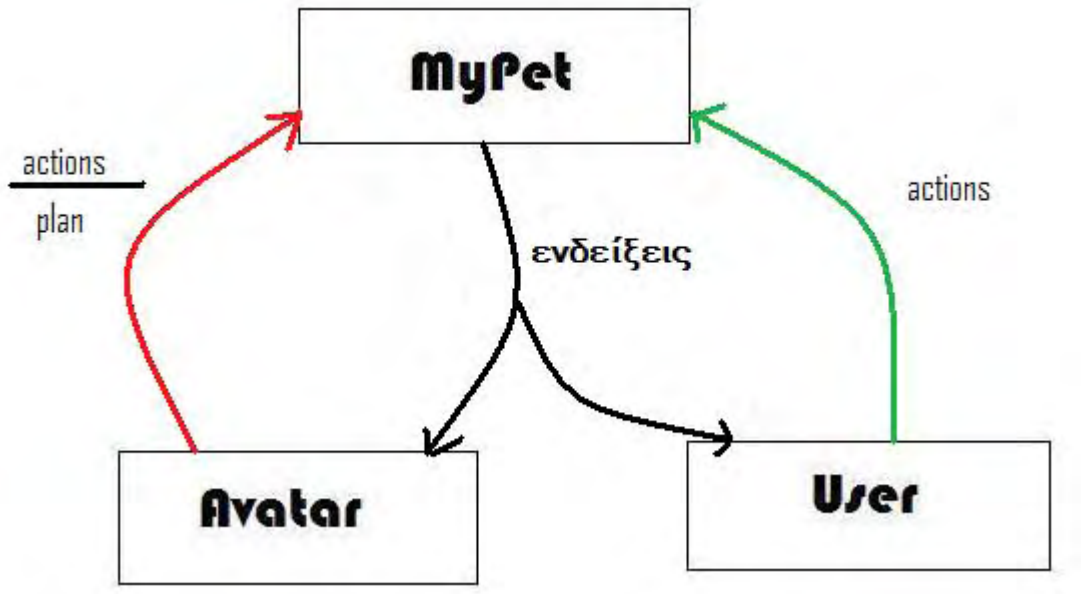
Ο χρήστης κάθε 30 δευτερόλεπτα πρέπει να προσφέρει φάρμακο στο MyPet.

Φάρμακο (μειώνει happiness -0.5, αυξάνει health +2)

Ο User μπορεί να ξεχάσει να αλληλεπιδράσει με το MyPet όταν Busy>3.

Αν Busy > 3 τότε κατά 60% αλληλεπιδρά με MyPet και κατά 40% ξεχνά να αλληλεπιδράσει

Αν Busy < 3 τότε κατά 90% αλληλεπιδρά με MyPet και κατά 10% ξεχνά να αλληλεπιδράσει


MyPet
actions
plan
ενδείξεις
actions
Avatar
User

3.2 Αντιδραστικοί Πράκτορες

3.2.1 Αρχιτεκτονική

Τα MyPet είναι σχεδιασμένοι σαν αντιδραστικοί πράκτορες. Δηλαδή έχουμε ανυπαρξία της εσωτερικής αναπαράστασης του κόσμου στα ρομπότ μας.

Η συμπεριφορά τους είναι βασισμένη σε μια φιλοσοφία ερεθίσματος/αντίδρασης(stimulus/response). Δηλαδή παίρνουν δεδομένα από το περιβάλλον(αντίληψη) και σύμφωνα με του κανόνες λειτουργίας αποφασίζουν ποια θα είναι η επόμενη ενέργεια τους. Δεν έχουν μνήμη (εικόνα 6).

3.2.2 Ψευδοκώδικας Λειτουργίας

Function SimpleReflexAgent (environmentstate) returns action

Static: Rules

Begin

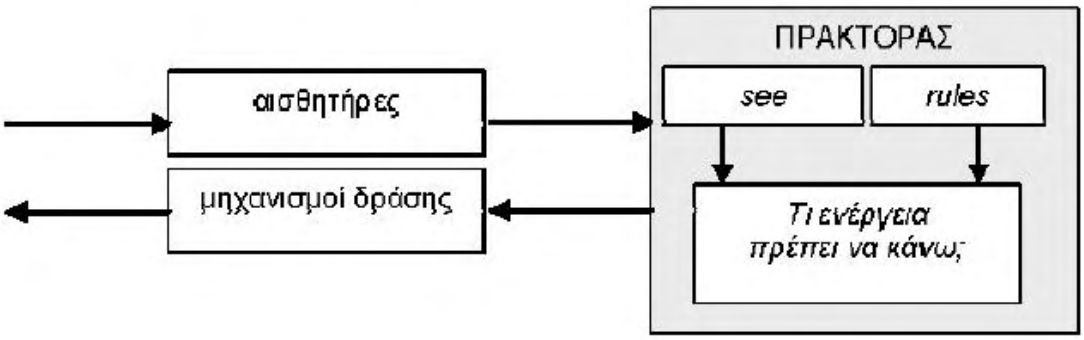
Percept<- see(environmentstate)

Rule<-match(percept,rules)

Action<-apply(rule)

Return action

End


ΠΡΑΚΤΟΡΑΣ
αισθητήρες
μηχανισμοί δράσης
see
rules
Τι ενέργεια
πρέπει να κάνω;

3.3.2 Υλοποίηση των νημάτων στην Java

Για να ξεκινά ένα νήμα από ένα αντικείμενο στη Java, θα πρέπει η τάξη από την οποία προέρχεται να υλοποιεί την μέθοδο run(). Στην ορολογία της Java τα αντικείμενα αυτά λέγονται Runnable objects. Η μέθοδος run() του αντικειμένου θα είναι η «πύλη» από την οποία θα ξεκινά ένα νέο νήμα εκτέλεσης. Το νήμα εκτέλεσης θα ξεκινά με την μέθοδο run() και θα τερματίζεται όταν τερματίζεται η μέθοδος run().

Υπάρχουν δύο τρόποι για να κάνουμε μια τάξη στη Java να υλοποιήσει την μέθοδο run():

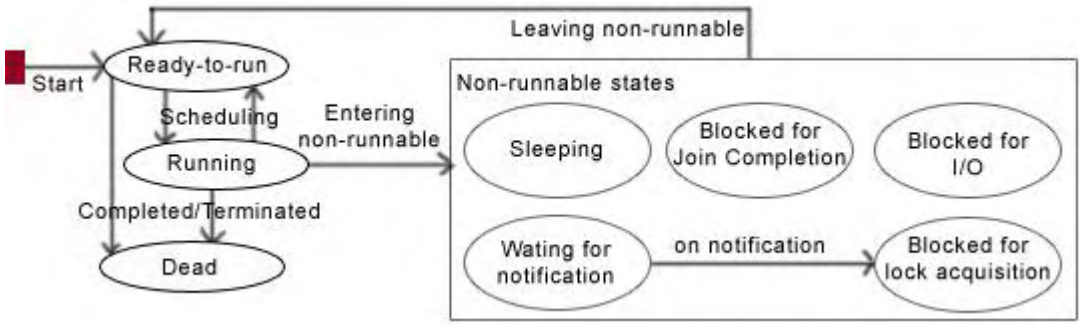
1. Ο πρώτος τρόπος είναι να κάνουμε την τάξη υποτάξη της τάξης Thread. Αυτή η τάξη παρέχει μια κενή μέθοδο run() και λογικά θα πρέπει να την ορίσουμε και πάλι στην υποτάξη υπερβαίνοντας την κενή run() της υπερτάξης. Η τάξη Thread υλοποιεί την διασύνδεση (interface) Runnable που πρέπει να υλοποιούν όλα τα αντικείμενα από τα οποία ξεκινούνε νέα νήματα εκτέλεσης. Η υποχρέωση που προκύπτει από την υλοποίηση του Runnable είναι ο ορισμός της μεθόδου run().

2. Ο δεύτερος τρόπος είναι η απευθείας υλοποίηση της διασύνδεσης Runnable. Σε αυτή τη περίπτωση θα πρέπει να οριστεί εξαρχής η μέθοδος run(). Αυτή η μέθοδος προτιμάται σε περιπτώσεις που η τάξη που ορίζουμε έχει ήδη μια άλλη υπερτάξη.

3.3.3 Υλοποιώντας την διασύνδεση Runnable

Η δεύτερη μέθοδος με την υλοποίηση της διασύνδεσης Runnable, όπως ήδη ανέφερα, είναι ο τρόπος με τον οποίο μπορούμε να κάνουμε μια τάξη να είναι τάξη από την οποία παράγονται ενεργά αντικείμενα και ας έχει ήδη μια άλλη υπερτάξη.

Σε αυτή τη περίπτωση δεν γίνεται να χρησιμοποιήσουμε την πρώτη μέθοδο γιατί η Java δεν υποστηρίζει την λεγόμενη πολλαπλή κληρονομικότητα (multiple inheritance), δηλαδή στη Java δεν γίνεται μια τάξη να επεκτείνει περισσότερες από μια τάξεις. Γι’ αυτό και εμείς χρησιμοποιούμε την Runnable καθώς στο


Leaving non-runnable
Start
Ready-to-run
Scheduling
Running
Entering
non-runnable
Completed/Terminated
Dead
Non-runnable states
Sleeping
Blocked for
Join Completion
Blocked for
I/O
Wating for
notification
on notification
Blocked for
lock acquisition

Σε κατάσταση εκτέλεσης (**Running)**

Όταν ο *χρονοπρογραμματιστής* των threads (***thread scheduler***) επιλέγει το thread από το *Runnable thread's pool*, το thread ξεκινά να εκτελείται.

Σε κατάσταση Αναμονής (Wait)/ Εμπλοκής (Blocked) / Ύπνου (**Sleeping)**

Σε αυτές τις καταστάσεις το thread θεωρείται «εν ζωή» αλλά όχι σε κατάσταση *έτοιμου προς εκτέλεση* (runnable). Το thread μετέρχεται στην κατάσταση αυτή εξαιτίας είτε μετά από την κλήση μίας μεθόδου wait() είτε μίας μεθόδου sleep() είτε περιμένοντας για κάποιον πόρο εισόδου/εξόδου.

Σε κατάσταση Αναμονής για Ειδοποίηση Αφύπνισης (Waiting for Notification)

Ένα thread περιμένει ειδοποίηση από ένα άλλο thread.

Σε κατάσταση Εμπλοκής για ολοκλήρωση της Συνεύρεσης (Blocked for joint completion)

Το thread μπορεί να εισέλθει σε αυτήν την κατάσταση εξαιτίας της αναμονής για την ολοκλήρωση ενός άλλου thread.

Σε κατάσταση θανάτου (Dead)

Όταν το thread ολοκληρώνει την εκτέλεση του μετά το πέρας της μεθόδου run(), εισέρχεται σε αυτήν την κατάσταση. Μετά από αυτήν την κατάσταση δεν μπορεί να ξεκινήσει ξανά, εάν δηλαδή η μέθοδος start() κληθεί σε ένα τέτοιο thread θα συμβεί ένα runtime exception.

3.3.5 Wait, Sleep, Interrupts, Synchronized

Μέθοδος wait()

Η μέθοδος **wait()** που είναι μέθοδος της τάξης Object (υπερτάξης όλων των τάξεων), προκαλεί την αναμονή στο νήμα εκτέλεσης που την καλεί, μέχρις ότου ένα άλλο νήμα εκτέλεσης καλέσει την μέθοδο **notify()** ή την μέθοδο **notifyAll()**.

Το νήμα εκτέλεσης που καλεί την wait() θα πρέπει να έχει το lock του αντικειμένου. Με άλλα λόγια η wait() πρέπει να καλείται μόνο μέσα από synchronized μεθόδους ή synchronized τμήματα κώδικα. Όταν κληθεί η wait() το νήμα απελευθερώνει το lock και περιμένει μέχρι κάποιο άλλο νήμα (που προφανώς πήρε το lock) να καλέσει την notify() ή την notifyAll(). Το νήμα εκτέλεσης τότε περιμένει μέχρις ότου ανακτήσει και πάλι την ιδιοκτησία της κλειδαριάς, οπότε και συνεχίζει την εκτέλεσή του.

Η μέθοδος wait() μπορεί να προκαλέσει την μη ελεγχόμενη εξαίρεση IllegalMonitorException εφόσον το αντικείμενο που καλέσει την wait() δεν έχει την ιδιοκτησία του lock στο αντικείμενο. Επίσης μπορεί να προκαλέσει την ελεγχόμενη (πρέπει να δηλωθεί ότι προκαλείται ή να γίνει ο χειρισμός του) εξαίρεση InterruptedException εάν κάποιο άλλο νήμα εκτέλεσης διακόψει (με την κλήση της interrupt) το εν λόγω νήμα εκτέλεσης ή διακοπεί εξαιτίας κάποιου λάθους η εκτέλεση του νήματος.

Όταν ένα νήμα αφήσει το lock με την wait(), κάποιο από τα υπόλοιπα νήματα που περίμεναν για το lock θα εισέλθει στο κρίσιμο τμήμα του (critical section). Το νήμα που εκτέλεσε την wait() θα περιμένει μέχρις ότου κάποιο άλλο νήμα εκτελέσει την **notify()** ή την **notifyAll()**. Η διαφορά των δύο μεθόδων είναι ότι η notifyAll() ειδοποιεί όλα τα νήματα που έχουν εκτελέσει την wait() και βρίσκονται στη μια και μοναδική λίστα αναμονής (waiting list) του αντικειμένου, ενώ η notify() επιλέγει με τυχαίο τρόπο ένα νήμα και του δίνει το lock για να συνεχίσει στο κρίσιμο τμήμα του.

Αναστολή Εκτέλεσης με τη μέθοδο Sleep()

Η μέθοδος Thread.sleep αναστέλει την εκτέλεση του νήματος για δεδομένο χρονικό διάστημα. Με αυτό το τρόπο ένα νήμα παραχωρεί χρόνο εκτέλεσης σε άλλα νήματα της εφαρμογής ή σε άλλες εφαρμογές. Η μέθοδος sleep μπορεί επίσης να χρησιμοποιηθεί για ρύθμιση της ταχύτητας εκτέλεσης, όπως φαίνεται στο παρακάτω παράδειγμα, ή για αναμονή εκτέλεσης κάποιου άλλου νήματος που πιθανόν να είναι πιο απαιτητικό σε χρόνο εκτέλεσης.

Το χρονικό διάστημα στη sleep ορίζεται σε milliseconds ή σε nanoseconds. Ο ακριβής χρονισμός εξαρτάται από την υλοποίηση της JVM και το λειτουργικό σύστημα. Επίσης η περίοδος αναστολής μπορεί να διακοπεί από κάποιο σήμα προς το νήμα. Σε κάθε περίπτωση η διάρκεια εκτέλεσης της sleep είναι προσεγγιστική.

Υποσημείωση: Ότι στο main της run() δηλώνεται throws InterruptedException. Αυτό σημαίνει οτι όταν ένα νήμα προσπαθήσει να διακόψει τη sleep η διακοπή αυτή απορρίπτεται. Αυτό συμβαίνει γιατί στην εφαρμογή μας δεν έχουμε ορίσει κάποιο άλλο νήμα που να προκαλεί διακοπή. Επομένως η εφαρμογή δεν ενδιαφέρεται να συλλάβει το InterruptedException.

Διακοπές (Interrupts)

Μια *διακοπή* (*interrupt)* είναι ένα σήμα προς το νήμα ότι πρέπει να σταματήσει αυτό που κάνει και να χειριστεί τη διακοπή. Ο ακριβής χειρισμός της διακοπής είναι ζήτημα του προγραμματιστή, αλλά είναι συνήθως ο τερματισμός της εκτέλεσής του νήματος. Αυτή είναι η χρήση που δείχνουμε εδώ.

'Ενα νήμα στέλνει σήμα διακοπής με την εφαρμογή της μεθόδου interrupt στο αντικείμενο Thread που αντιστοιχεί στο προς διακοπή νήμα. Το νήμα που παραλαμβάνει τη διακοπή πρέπει να τη διαχειριστεί κατάλληλα.

Διαχείριση Διακοπής

Η διαχείριση διακοπής εξαρτάται από το τι κάνει το νήμα. Αν το νήμα εκτελεί μεθόδους που μπορούν να διακοπούν, δηλαδή προκαλούν InterruptedException, τότε η σύλληψη του InterruptedException απλά σημαίνει επιστροφή από τη μέθοδο run.

Μέθοδος Συνεύρεσης (Join)

Η μέθοδος join αναγκάζει ένα νήμα να περιμένει το τερματισμό άλλου νήματος. Έστω t ένα αντικείμενο Thread που εκτελείται συχρονικά με ένα άλλο νήμα. Αν το δεύτερο νήμα φτάσει στη κλήση t.join(); τότε το δεύτερο νήμα θα αναστείλει την εκτέλεσή του μέχρι το νήμα t να τερματίσει. Εκδοχές της join επιτρέπουν τον καθορισμό χρόνου αναμονής. Όμως, όπως και στη sleep, ο χρονισμός δεν είναι απόλυτα ακριβής. Η join, όπως και η sleep, όταν δέχεται διακοπή τερματίζει με InterruptedException.

Synchronized

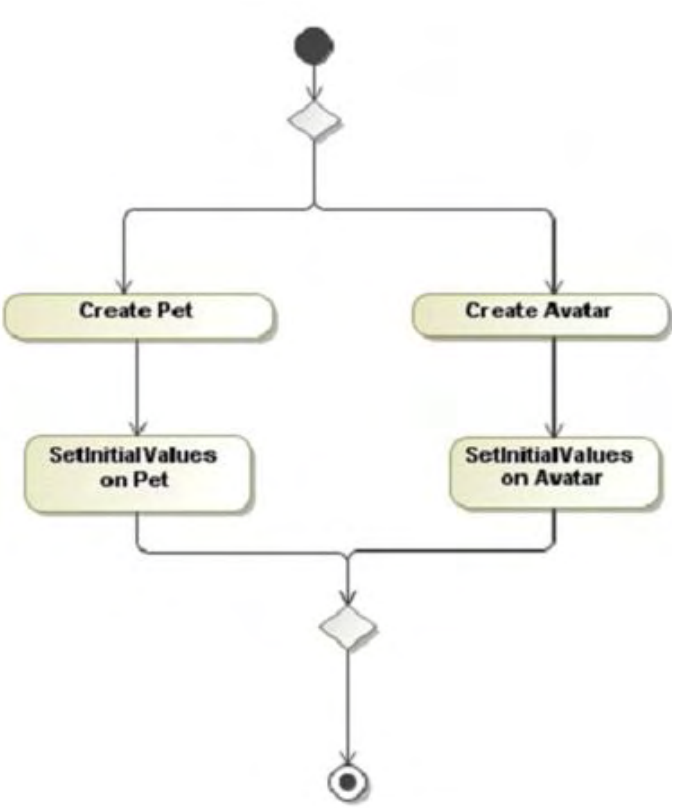
Η Java υποστηρίζει τον αμοιβαίο αποκλεισμό με χρήση της δεσμευμένης λέξης **synchronized**, που μπορεί να χρησιμοποιηθεί είτε για τον συγχρονισμό σε ολόκληρες μεθόδους είτε και για το συγχρονισμό σε συγκεκριμένα τμήματα κώδικα (code blocks).

Η δήλωση μιας synchronized μεθόδου έχει ως αποτέλεσμα την αποκλειστική εκτέλεση της σε σχέση με όλες τις άλλες synchronized μεθόδους ή synchronized τμήματα κώδικα (στο ίδιο αντικείμενο), και συντάσσεται ως εξής:

προσδ.-πρόβασης **synchronized** τύπος όνομα (λίστα-παραμέτρων)
{

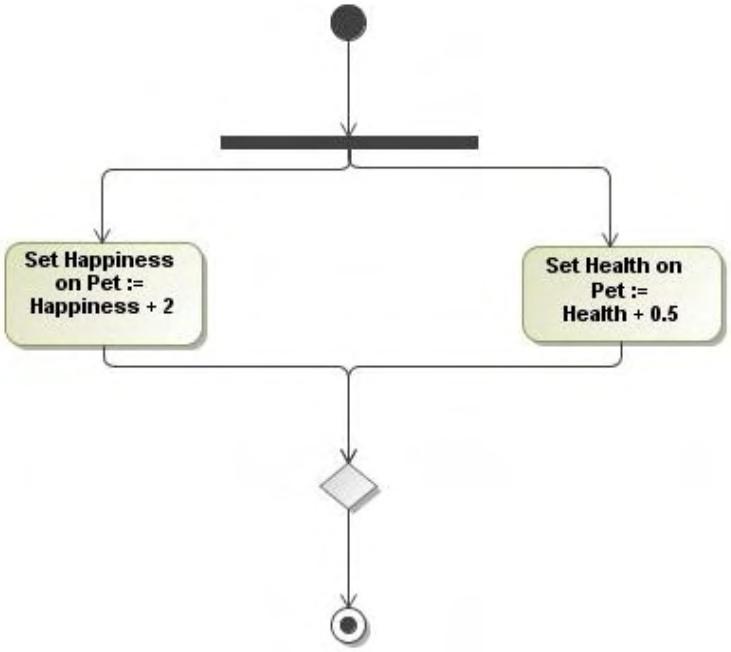
...
}

Η δήλωση ενός τμήματος κώδικα ως synchronized γίνεται με το ακόλουθο συντακτικό και ο κώδικας θα εκτελεστεί αποκλειστικά για

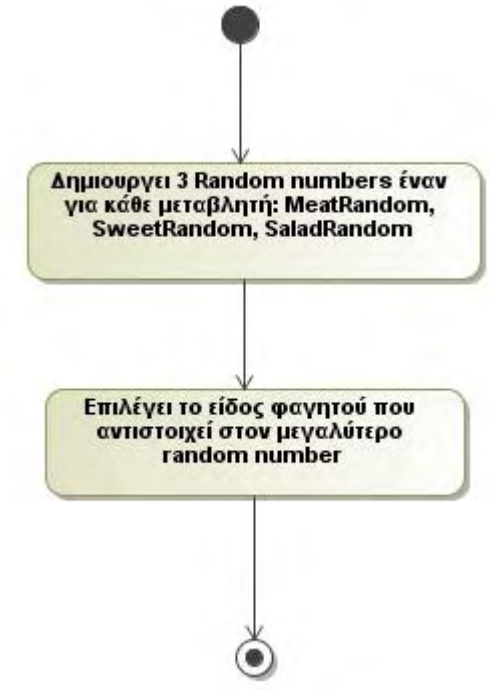


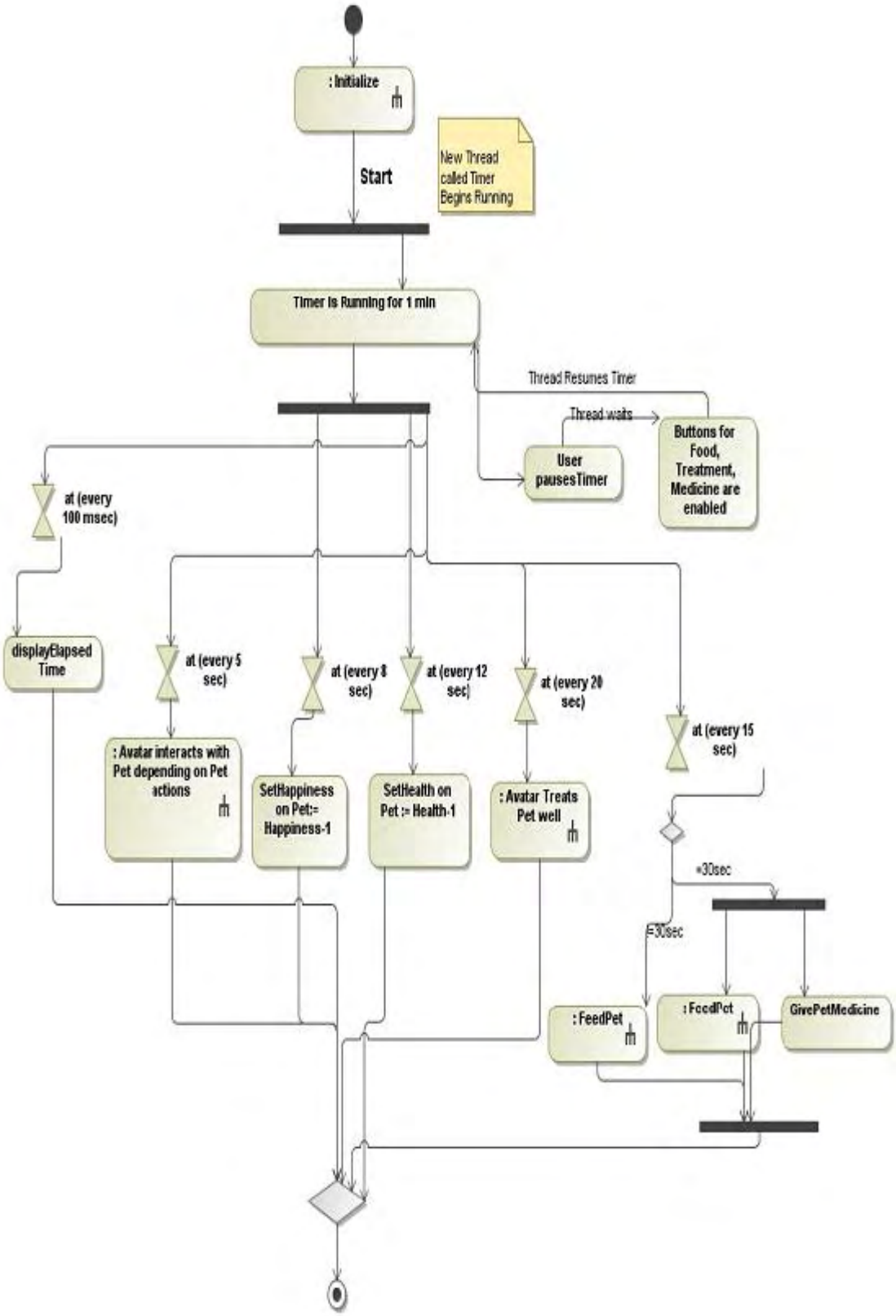
Στην αρχή καλείται η Initialize με την οποία δημιουργούνται δύο νέα στιγμιότυπα των κατασκευαστικών μεθόδων MyPet και Avatar.

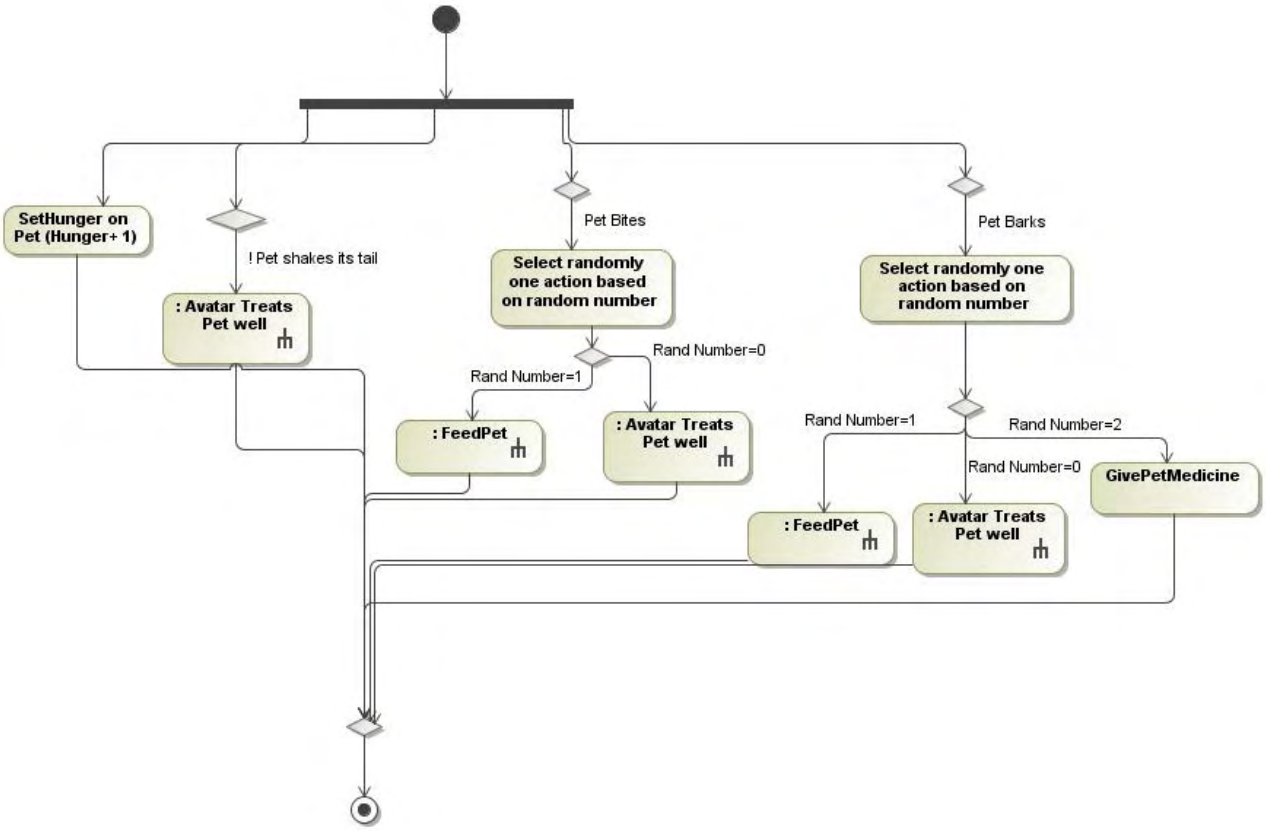
Έτσι αρχικοποιούνται οι τιμές των χαρακτηριστικών για τις 2 οντότητες του προγράμματος μας. Χρησιμοποιώ τυχαίες τιμές στην αρχικοποίηση κάνοντας χρήση της java.util.Random.

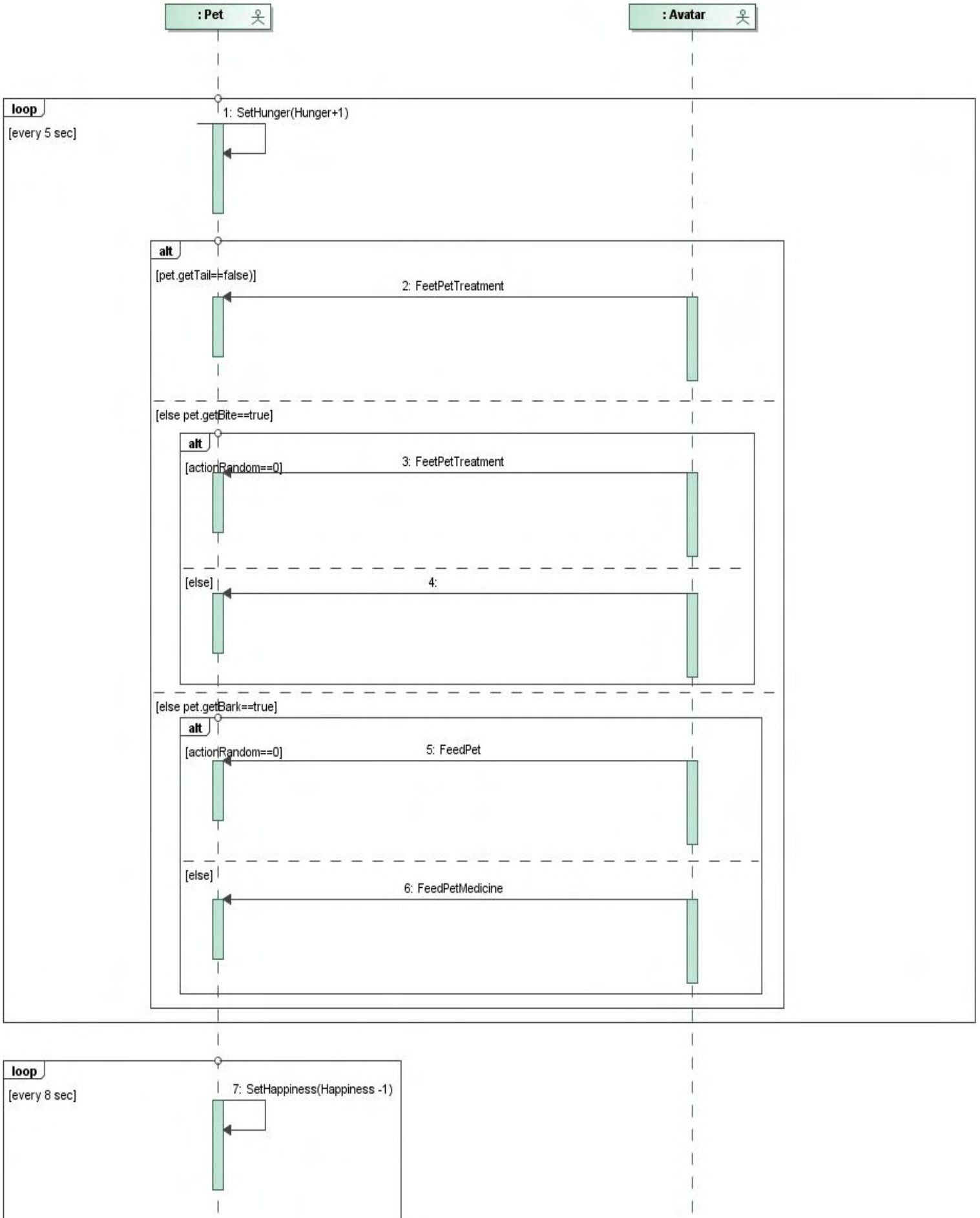


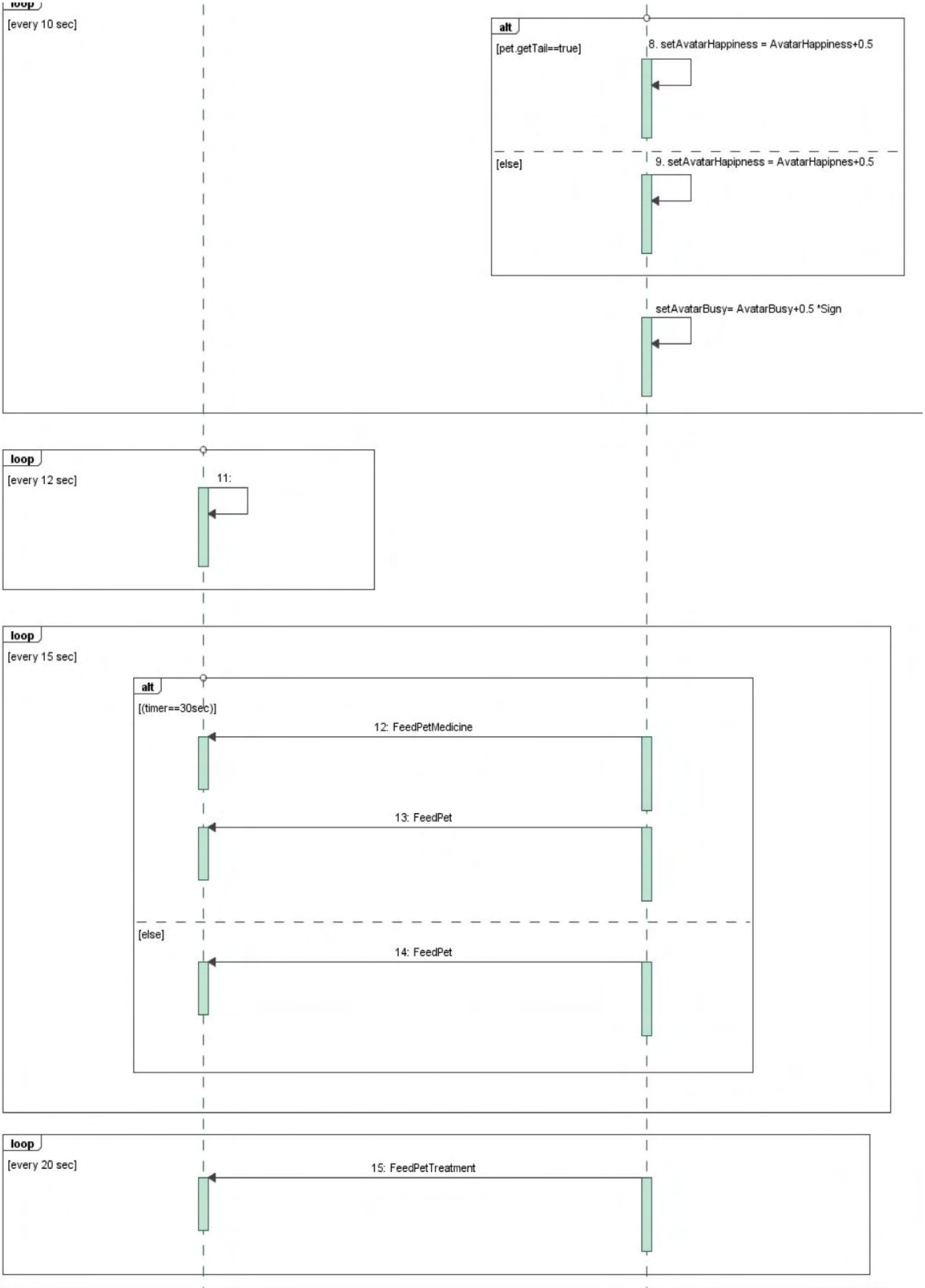
Όταν ο Avatar αλληλεπιδρά με καλή μεταχείριση για το MyPet, αυξάνει την ευτυχία του MyPet κατά 2 και την υγεία του κατά 0,5




: Initialize
Start
New Thread called Timer Begins Running
Timer is Running for 1 min
Thread Resumes Timer
Thread waits
User pausesTimer
Buttons for Food, Treatment, Medicine are enabled
at (every 100 msec)
displayElapsed Time
at (every 5 sec)
: Avatar interacts with Pet depending on Pet actions
at (every 8 sec)
SetHappiness on Pet:= Happiness-1
at (every 12 sec)
SetHealth on Pet := Health-1
at (every 20 sec)
: Avatar Treats Pet well
at (every 15 sec)
=30sec
<30sec
: FeedPet
: FeedPet
GivePetMedicine


SetHunger on Pet (Hunger+ 1)
! Pet shakes its tail
: Avatar Treats Pet well
Pet Bites
Select randomly one action based on random number
Rand Number=0
Rand Number=1
: FeedPet
: Avatar Treats Pet well
Pet Barks
Select randomly one action based on random number
Rand Number=1
Rand Number=2
Rand Number=0
GivePetMedicine
: FeedPet
: Avatar Treats Pet well


: Pet
: Avatar
loop
[every 5 sec]
1: SetHunger(Hunger+1)
alt
[pet.getTail==false)]
2: FeetPetTreatment
[else pet.getBite==true]
alt
[actionRandom==0]
3: FeetPetTreatment
[else]
4:
[else pet.getBark==true]
alt
[actionRandom==0]
5: FeedPet
[else]
6: FeedPetMedicine
loop
[every 8 sec]
7: SetHappiness(Happiness -1)


loop
[every 10 sec]
alt
[pet.getTail==true]
8. setAvatarHappiness = AvatarHappiness+0.5
[else]
9. setAvatarHapipness = AvatarHapipnes+0.5
setAvatarBusy= AvatarBusy+0.5 *Sign
loop
[every 12 sec]
11:
loop
[every 15 sec]
alt
[(timer==30sec)]
12: FeedPetMedicine
13: FeedPet
[else]
14: FeedPet
loop
[every 20 sec]
15: FeedPetTreatment


MyPetStart
-pet : MyPet
-avatar : Avatar
-startTime : long
-updater : Thread
-isRunning : boolean = false
~round : boolean = false
~done : boolean = false
-startmil : long
-paused : boolean = false
-stopped : boolean = false
-timeout : boolean = false
-jCheckInput : JCheckBox = null
-startStopButton : JButton = null
-StopButton : JButton = null
-jTextHunger : JTextField = null
-jTextHealth : JTextField = null
-jTextHappiness : JTextField = null
-jButtonNewValuesPet : JButton = null
-jButtonMeatFood : JButton = null
-jButtonSaladFood : JButton = null
-jButtonsSweetFood : JButton = null
-jButtonTreatment : JButton = null
-jButtonMedicine : JButton = null
-jComboHunger : JComboBox = null
-jComboHealth : JComboBox = null
-jComboHappiness : JComboBox = null
-jComboHappinessTail : JComboBox = null
-jTextHappinessAvatar : JTextField = null
-jTextBusyAvatar : JTextField = null
-jButtonNewValuesAvatar : JButton = null
...
<<constructor>>+MyPetStart()
-initialize() : void
+run() : void
-displayElapsedTime(elapsedTime : long) : void
+actionPerform(ae : ActionEvent) : void
+UserTreatPet() : void
+TreatPet() : void
+GivePetMedicine() : void
+FeedPet() : void
+FeedPetMeat() : void
+FeedPetSalad() : void
+FeedPetSweet() : void
+FeedPetMedicine() : void
+FeedPetTreatment() : void
+SetPetHappiness(happiness : double) : void
+SetPetHealth(health : double) : void
<<getter>>-getJTextHunger() : JTextField
<<getter>>-getJTextHealth() : JTextField
<<getter>>-getJTextHappiness() : JTextField
<<getter>>-getJButtonNewValuesPet() : JButton
<<getter>>-getStartStopButton() : JButton
<<getter>>-getJButtonMeatFood() : JButton
<<getter>>-getJButtonSaladFood() : JButton
<<getter>>-getJButtonsSweetFood() : JButton
<<getter>>-getJButtonTreatment() : JButton
<<getter>>-getJButtonMedicine() : JButton
<<getter>>-getJCheckInput() : JCheckBox
<<getter>>-getStopButton() : JButton
<<getter>>-getJComboHunger() : JComboBox
<<getter>>-getJComboHealth() : JComboBox
<<getter>>-getJComboHappiness() : JComboBox
<<getter>>-getJComboHappinessTail() : JComboBox
<<getter>>-getJTextHappinessAvatar() : JTextField
<<getter>>-getJTextBusyAvatar() : JTextField
<<getter>>-getJButtonNewValuesAvatar() : JButton
+main(arg : String"[]") : void
...
MyPet
+Hunger : double
+Health : double
+Happiness : double
+Tail : boolean
+Bite : boolean
+Bark : boolean
+Dead : boolean
+BarkHunger : double
+BarkHealth : double
+BarkHappiness : double
+TailHappiness : double
<<constructor>>+MyPet()
#finalize() : void
+SetInitialValues() : void
+SetPetStatus(hunger : double, health : double, happiness : double) : void
+Bark() : void
+Bark(hunger : double, health : double, happiness : double) : void
<<getter>>+getBark() : boolean
+Tail() : void
+Tail(happiness : double) : void
<<getter>>+getTail() : boolean
<<setter>>+setTail(tail : boolean) : void
+Bite(hunger : double, happiness : double) : void
<<getter>>+getBite() : boolean
<<setter>>+setBite(bite : boolean) : void
+Dies(health : double) : void
<<setter>>+setDead(dead : boolean) : void
<<getter>>+getHappiness() : double
<<setter>>+setHappiness(happiness : double) : void
<<getter>>+getHealth() : double
<<setter>>+setHealth(health : double) : void
<<getter>>+getHunger() : double
<<setter>>+setHunger(hunger : double) : void
<<getter>>+getBarkHappiness() : double
<<setter>>+setBarkHappiness(barkHappiness : double) : void
<<getter>>+getBarkHealth() : double
<<setter>>+setBarkHealth(barkHealth : double) : void
<<getter>>+getBarkHunger() : double
<<setter>>+setBarkHunger(barkHunger : double) : void
<<getter>>+getTailHappiness() : double
<<setter>>+setTailHappiness(tailHappiness : double) : void
Avatar
+Busy : double
+Happiness : double
<<constructor>>+Avatar()
+selectFood() : String
<<getter>>+getHappiness() : double
<<setter>>+setHappiness(happiness : double) : void
<<getter>>+getBusy() : double
<<setter>>+setBusy(busy : double) : void
+SetInitialValues() : void
StartMain
+main(args : String"[]") : void

3.5 Ιδιαίτερα προγραμματιστικά σημεία που χρησιμοποιήθηκαν στον προγραμματισμό για τις κυριότερες λειτουργίες

3.5.1 Στην κλάση Avatar

Η **selectFood** χρησιμοποιείται από τον χρήστη avatar για να επιλέξει τυχαία το φαγητό που θα δώσει στο pet.

Ο αλγόριθμος δημιουργεί τρεις τυχαίους αριθμούς για meat, salad και sweet και ανάλογα με το ποιος είναι ο μέγιστος επιστρέφει το αντίστοιχο είδος φαγητού.

```
public String selectFood() {
String food="";
float meatRandom = new Random().nextFloat();
float sweetRandom = new Random().nextFloat();
float saladRandom = new Random().nextFloat();
/* Επιστρεφει την τιμη που ειναι μεγιστη*/
if (meatRandom>=sweetRandom && meatRandom>saladRandom) {
        food= "meat";
}
if (sweetRandom>meatRandom && sweetRandom>saladRandom){
        food= "sweet";
}
if (saladRandom>=meatRandom && saladRandom>=sweetRandom){
        food= "salad";
}
return food;
}
```

Εικόνα 8 Γραφικό περιβάλλον του MyPetStart

Η οθόνη έχει ένα Start/Pause button, στο οποίο κάνοντας κλικ ξεκινά ο timer, και το Stop button με το οποίο σταματά.

Στην **initialize** δημιουργώ το αντικείμενο pet της κλάσης MyPet, αρχικοποιώντας τις ιδιότητες του αντικειμένου του (Hunger, Health και Happiness). Η αρχικοποίηση γίνεται καλώντας τη SetInitialValues για να θέσει τυχαίες τιμές στα 3 χαρακτηριστικά: Hunger, Health και Happiness. Υπάρχουν 3 text boxes που εμφανίζουν τις τρέχουσες τιμές που έχουν κατά τη διάρκεια του προγράμματος. Με το button "Αρχικοποίηση τιμών" ο χρήστης μπορεί να σετάρει τα 3 χαρακτηριστικά με τυχαίες τιμές καλώντας τη SetInitialValues.

Στο κάτω τμήμα του παραθύρου υπάρχουν checkboxes που σετάρουν την ένταση των 3 χαρακτηριστικών για να ρυθμίσουν τις συνθήκες στις ενέργεις Bark και Tail. Αρχικά έχουν όλα τιμή 3 όπως έχει ζητηθεί από τα δεδομένα του προβλήματος.

Η εικόνα στα αριστερά του παραθύρου παρουσιάζει την κατάσταση του αντικειμένου pet ανάλογα με την ενέργεια της οποίας η συνθήκη ικανοποιείται: (Bark, Tail, Bite, Dies). Όταν το pet τρέχει την ενέργεια Bark ακούγεται ηχητικό μήνυμα.

Επειδή οι συνθήκες σε κάποιες ενέργειες αλληλεπικαλύπτονται δίνεται προτεραιότητα σε αυτήν που έχει τις λιγότερες συνθήκες: π.χ. Dies έναντι των άλλων.

Η εικόνα στα δέξια του παραθύρου είναι του Avatar και από πάνω της είναι τα δύο χαρακτηριστικά του Avatar: Busy και Happiness.

```
private void initialize() {.....

                pet= new MyPet();

                jTextHunger.setText(Double.toString(pet.Hunger));

                jTextHealth.setText(Double.toString(pet.Health));

                jTextHappiness.setText(Double.toString(pet.Happiness));


                pet.setBarkHunger(3);

                pet.setBarkHealth(3);

                pet.setBarkHappiness(3);
```

```
                pet.setTailHappiness(3);


                jComboHunger.setSelectedIndex((int)pet.getBarkHunger());
                jComboHealth.setSelectedIndex((int)pet.getBarkHealth());
                jComboHappiness.setSelectedIndex((int)pet.getBarkHappiness());
                jComboHappinessTail.setSelectedIndex((int)pet.getTailHappiness());


                avatar= new Avatar();
                jTextBusyAvatar.setText(Double.toString(avatar.Busy));
                jTextHappinessAvatar.setText(Double.toString(avatar.Happiness));
....
}
```

Η **windowClosing** λειτουργεί ως garbage collector, διασφαλίζοντας ότι το thread θα σταματά ακόμη και όταν ο χρήστης πατήσει το close button του παραθύρου προτού το thread ολοκληρώσει την εκτέλεση του. Χωρίς αυτόν τον έλεγχο, το thread μπορεί να τρέχει ακόμη και όταν ο χρήστης έχει πατήσει το close και η εφαρμογή δείχνει ότι έχει κλείσει, διότι σε περίπτωση που δεν έχουν παρέλθει τα 60 δευτερόλεπτα ή αν ο χρήστης δεν έχει πατήσει το stop to thread παραμένει στον task manager alive.

```
public void windowClosing(WindowEvent event) {
                if (updater!=null)
                {
                        if (paused)
                                {
                                updater.interrupt();
                                try
                                {
                                        updater.join();         // Wait for updater to finish
                                }
                                catch(InterruptedException ie)
{System.out.println("stopped2");}
                                } else
                                {
```

```
                updater.interrupt();
            }
        }
        System.exit(0);
    }
```

Η **setButtonsDefaultColor** χρησιμοποιείται για να επαναφέρει το χρώμα των buttons που γίνεται magenda, όταν ο avatar αλληλεπιδρά με το Pet.

```
private void setButtonsDefaultColor() {
        jButtonMeatFood.setBackground(defaultButtonColor);
        jButtonSaladFood.setBackground(defaultButtonColor);
        jButtonsSweetFood.setBackground(defaultButtonColor);
        jButtonTreatment.setBackground(defaultButtonColor);
        jButtonMedicine.setBackground(defaultButtonColor);
    }
```

Η εκτέλεση του thread ξεκινά όταν κληθεί η μέθοδος **start()**.

Η μεταβλητή **paused** σετάρεται όταν πατηθεί το button "Pause". Όσο είναι η τιμή false τότε τρέχει ο timer και οι αλληλεπιδράσεις με το pet, διαφορετικά γίνεται sleep().

```
        while (isRunning)
        {
            while  (!paused)
            {...
```

Κατά τη διάρκεια της εκτέλεσης του thread καλείται η **displayElapsedTime** κάθε 100 milliseconds για να εμφανίσει τα δευτερόλεπτα που έχουν περάσει.

```
Thread.sleep(100);


jButtonMeatFood.setEnabled(false);
jButtonSaladFood.setEnabled(false);
jButtonsSweetFood.setEnabled(false);
jButtonMedicine.setEnabled(false);
jButtonTreatment.setEnabled(false);
displayElapsedTime(elapsedTime);
```

Η μεταβλητή **calcmil** δείχνει σε ποιο χιλιοστό του δευτερολέπτου βρίσκεται κάθε φορά ο timer. Η **calcmil** αποκτά κάθε φορά την τιμή που δηλώνει το κλάσμα του second (σε milliseconds) που τρέχει στον timer, ώστε ο timer να ενημερώνει τις αντίστοιχες χαρακτηριστικές του MyPet. Σε κάθε δευτερόλεπτο του timer, η calcmil παίρνει τιμές μεταξύ 0 και 1000 (msec).

Αυτή η μεταβλητή χρησιμοποιείται, διότι έχω επιλέξει το thread του timer να αναστέλλει την εκτέλεση του για 100 ms, με τη μέθοδο sleep(100), εφόσον θέλω ο timer να εμφανίζει το χρόνο που έχει παρέλθει κάθε 100 millisec, και εφόσον οι έλεγχοι στο pet γίνονται μόνο στα πρώτα 100 milliseconds του current second, ώστε να ενημερώνονται οι χαρακτηριστικές του εφόσον ικανοποιούν τις συνθήκες του προβλήματος.. Για παράδειγμα στο 15ο δευτερόλεπτο, υπάρχουν 10 διαστήματα των 100 millisec, αλλά για να γίνει ένα και μόνο σετάρισμα στις τιμές του pet, χρησιμοποιώ τη μεταβλητή calcmil, πραγματοποιώντας έλεγχο ώστε αυτή να έχει τιμή μέσα στα πρώτα 100 millisec (calcmil<100). Εναλλακτικά, θα μπορούσαμε αντί για τη συνθήκη calcmil<100 να βάλουμε calcmil>900, έχοντας το ίδιο αποτέλεσμα.

```
long calcmil = mils%1000;
    if ((sec%15 == 0) & (sec>0) )
                                {
                                        if (calcmil<100)
....
```

Η **actionPerform** εκτελείται, όταν πατηθεί το Start/Pause button.

Εάν ο thread updater (και ο timer) είναι σε κατάσταση εκτέλεσης, τότε κάνει pause ή start ανάλογα με την περίπτωση σετάροντας τη μεταβλητή paused, διαφορετικά εκκινεί την εκτέλεση του thread καλώντας την μέθοδο start();

Το πώς η μεταβλητή paused επιδρά στον timer εξαναγκάζοντας το thread να εισέλθει είτε σε κατάσταση sleep είτε εκτέλεσης (running) φαίνεται από το τμήμα του κώδικα στα try / catch exceptions της μεθόδου run().

```
//System.out.println("paused");
        if ((timeout) || (stopped))
        {
            timeout= false;
                                        Thread.currentThread().interrupt();
                                        startStopButton.setEnabled(true);
                                        /*if (stopped)
                                                updater2.interrupt();*/
        }
        Thread.sleep(Long.MAX_VALUE);
      } catch (InterruptedException e) {
        paused = false;
        //System.out.println("woke up");
        jButtonMeatFood.setEnabled(false);
                                jButtonSaladFood.setEnabled(false);
                                jButtonsSweetFood.setEnabled(false);
                                jButtonMedicine.setEnabled(false);
                                jButtonTreatment.setEnabled(false);
      }
-----------------------------------------------
public void actionPerform(ActionEvent ae)
            {
                    if (isRunning)
```

```
        {
                jButtonMeatFood.setEnabled(true);
                jButtonSaladFood.setEnabled(true);
                jButtonsSweetFood.setEnabled(true);
                jButtonMedicine.setEnabled(true);
                jButtonTreatment.setEnabled(true);
                jButtonNewValuesPet.setEnabled(true);
                jButtonNewValuesAvatar.setEnabled(true);
                paused =true;
                updater.interrupt();
        }
        else
        {
                /**/
                startTime= System.currentTimeMillis();
                startmil = java.util.concurrent.TimeUnit.MILLISECONDS.toMillis( startTime);

                updater= new Thread(this);
                updater.start();

                paused= false;
                stopped = false;
                isRunning= true;
                jButtonMeatFood.setEnabled(false);
                jButtonSaladFood.setEnabled(false);
                jButtonsSweetFood.setEnabled(false);
                jButtonMedicine.setEnabled(false);
                jButtonTreatment.setEnabled(false);
                jButtonNewValuesPet.setEnabled(false);
                jButtonNewValuesAvatar.setEnabled(false);
        }
}
```

Αφού πατηθούν τα buttons που αλληλεπιδρούν με το MyPet περνούν **2 sec** για να επανέλθει το χρώμα

```
if ((secToRestoreColor==2) & (calcmilToRestoreColor<100))
                {
                        setButtonsDefaultColor();
                }
```

Οι χαρακτηριστικές τιμές του MyPet **παίρνουν τιμές κάθε 5,8,12 sec**.

Κάθε 5 sec αλληλεπιδρά και User με το MyPet εκτός των παραπάνω προγραμματισμένων στιγμών και ανάλογα με τις ενέργειες του MyPet (Tail, Bark,Bite).

```
if (((sec%5 == 0) & (sec>0) & (calcmil<100)))
                {
                        pet.setHunger(pet.Hunger + 1);
                        jTextHunger.setText(Double.toString(pet.Hunger));

                        if (!pet.getTail()) {
                                startTimeToRestoreColor=System.currentTimeMillis();
                                FeedPetTreatment();
                        }
                        if (pet.getBite()) {
                                startTimeToRestoreColor=System.currentTimeMillis();
                                int actionRandom = new Random().nextInt(2);
                                if (actionRandom==0)
                                        FeedPetTreatment();
                                else
                                        FeedPet();
                        }
                        if (pet.getBark()) {
                                startTimeToRestoreColor=System.currentTimeMillis();
```

```
                    int actionRandom = new Random().nextInt(3);
                    if (actionRandom==0)
                        FeedPetTreatment();
                    else if (actionRandom==1)
                        FeedPet();
                    else
                        FeedPetMedicine();
                }
            }
            if ((sec%8 == 0) & (sec>0) & (calcmil<100))
            {
                pet.setHappiness(pet.Happiness - 1);
                jTextHappiness.setText(Double.toString(pet.Happiness));
            }
            if ((sec%12 == 0) & (sec>0) & (calcmil<100))
            {
                pet.setHealth(pet.Health - 1);
                jTextHealth.setText(Double.toString(pet.Health));
            }
```

Οι χαρακτηριστικές τιμές του User Avatar παίρνουν τιμές **κάθε 10 sec.**

```
if (((sec%10 == 0) & (sec>0) & (calcmil<100)))
            {
                double happinessRandom=0.5;
                double sign =1;
                if (!pet.getTail())
                    sign = -1;
                else
                    sign = 1;
                happinessRandom =happinessRandom*sign;
                avatar.setHappiness(avatar.Happiness + happinessRandom);

jTextHappinessAvatar.setText(Double.toString(avatar.Happiness));
```

```
                double busyRandom=0.5;
                boolean signus =new Random().nextBoolean();
                if (!signus)
                        busyRandom=-busyRandom;
                avatar.setBusy(avatar.Busy+ busyRandom);
                jTextBusyAvatar.setText(Double.toString(avatar.Busy));
        }
```

Η **round** χρησιμοποιείται για να δείξει ότι έχει τελειώσει ο timer.

```
if ((sec==59) & (calcmil<100))
                {


                }

                if ((sec==60) & (round==true)) {
                        round=false;
                isRunning=false;
                timeout= true;
                Thread.currentThread().interrupt();
                }
```

Κάθε **2 δευτερόλεπτα** το πρόγραμμα ελέγχει αν ικανοποιούνται οι συνθήκες που ενεργοποιούν τις ενέργειες του pet.

```
if ((sec%2==0) & (sec>0) & (calcmil<100) || ((sec<1) & (calcmil<100)))
                {
                        pet.Bark(pet.Hunger, pet.Health, pet.Happiness);
        pet.SetPetStatus(pet.Hunger, pet.Health, pet.Happiness);
        if (pet.Tail)
                jDogLabel.setIcon(new ImageIcon(getClass().getResource("/images/tail-dog.jpg")));
        else
                jDogLabel.setIcon(new ImageIcon(getClass().getResource("/images/dog.jpg")));
        if (pet.Dead)
```

```
                                        jDogLabel.setIcon(new
ImageIcon(getClass().getResource("/images/dead_dog.jpg")));
                                    else if (pet.Bite)
                                        jDogLabel.setIcon(new ImageIcon(getClass().getResource("/images/DogBite.jpg")));


                                    if (avatar.Happiness<2)
                                        jAvatarLabel.setIcon(new
ImageIcon(getClass().getResource("/images/avatar.jpg")));
                                    else
                                        jAvatarLabel.setIcon(new ImageIcon(getClass().getResource("/images/happy-
avatar.jpg")));
                                        }
```

Η **UserTreatPet** υλοποιεί την αλληλεπίδραση.

Ο User μπορεί να ξεχάσει να αλληλεπιδράσει 60% με το MyPet όταν Busy>3 και 90% όταν Busy<3

```
public void UserTreatPet()
{
        float f =new Random().nextFloat();
        int d = (int) (f*10.0);
        double p = ((float)d)/10.0;
        if (avatar.getBusy()>3) {
                if (p<0.6) {
                        TreatPet();
                }
        } else if (avatar.getBusy()<3) {
                if (p<0.8) {
                        TreatPet();
                }
        }
}
```

Η **TreatPet** υλοποιεί τη συναναστροφή καλή/κακή μεταχείρηση με το pet.

```
public void TreatPet()
{
	if (avatar.getHappiness()<2) {
		/* κακή μεταχείριση */
		pet.setHappiness(pet.Happiness - 2);
		pet.setHealth(pet.Health - 0.5);
	} else { /* καλή μεταχείριση */
		pet.setHappiness(pet.Happiness + 2);
		pet.setHealth(pet.Health + 0.5);
	}
	jTextHappiness.setText(Double.toString(pet.Happiness));
	jTextHealth.setText(Double.toString(pet.Health));
}
```

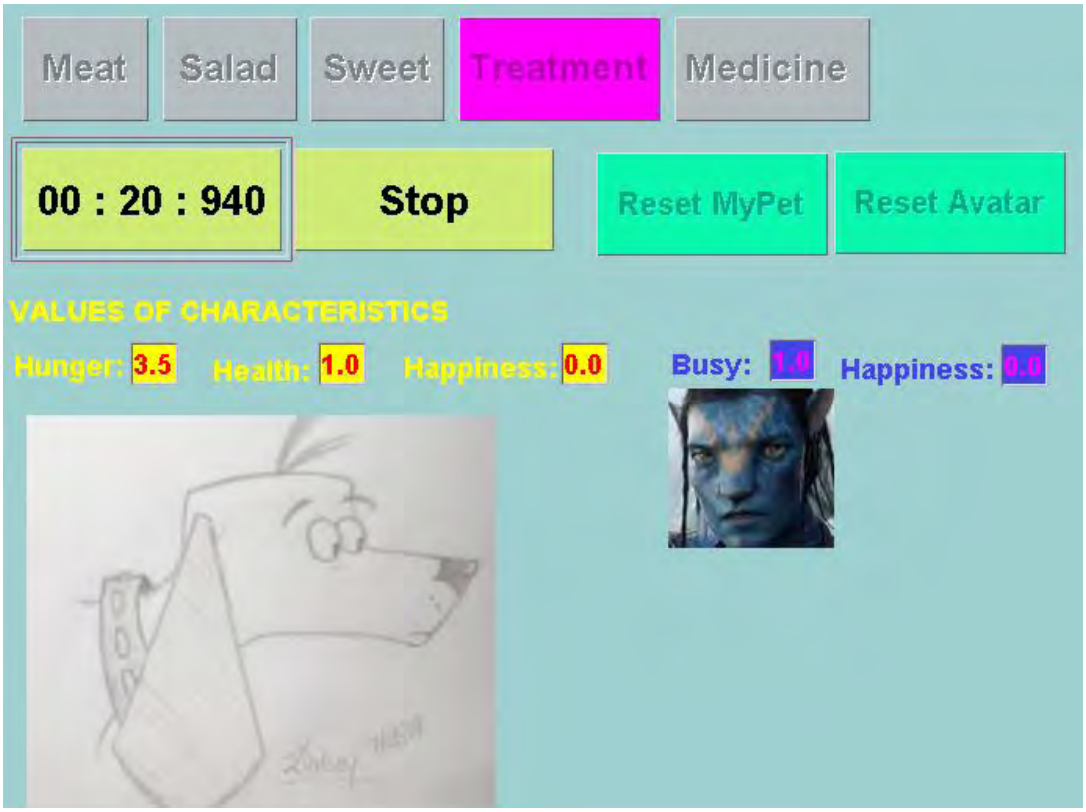
Οι μέθοδοι **SetPetHappiness** και **SetPetHealth** χρησιμοποιούν τη μεταβλητή done για να αποφευχθεί το deadlock που θα πραγματοποιούταν τη στιγμή που στο 30 sec προσπαθεί και η **FeedPetMedicine** και η **FeedPet** να προσπελάσουν τους κοινούς πόρους την ίδια χρονική στιγμή.

Κοινοί πόροι είναι οι **pet.Happiness, jTextHappiness, pet.Health** και **jTextHealth**.

```
public void SetPetHappiness(double happiness)
{
	if (!done) {
		done=true;
		pet.setHappiness(pet.Happiness+happiness);
		jTextHappiness.setText(Double.toString(pet.Happiness));
	}
}
public void SetPetHealth(double health)
{
```

Εικόνα 9 Printscreen στα 5 sec

Εικόνα 10 Printscreen στα 15 sec



Εικόνα 11 Printscreen στα 20 sec

4

Συμπεράσματα, Αποτίμηση και Μελλοντική δουλειά

4.1 Επίλογος

Το θέμα βέβαια που αρχίζει να ξεπροβάλλει είναι το πόσο και πότε μπορούν να γίνουν αποδεκτά από την κοινωνία και να ενσωματωθούν σε αυτή τα ρομπότ. Άσχετα με την ανθρώπινη αποδοχή ή όχι, τα ρομπότ με τεχνητή νοημοσύνη είναι βέβαιο πως θα επιδράσουν καταλυτικά στην κοινωνία μας. Οι άνθρωποι θα υιοθετήσουν συνήθειες που δεν μπορούμε σήμερα να προβλέψουμε! Αξιοσημείωτο είναι ότι ήδη τα ρομπότ μαθαίνουν συλλέγοντας στοιχεία από τους ανθρώπους. Τα στοιχεία συλλέγονται μέσω οπτικών, ακουστικών, απτικών(της αφής), κινητικών αισθητήρων. Αυτές οι μηχανές θα έχουν επιπλέον την ικανότητα να εντοπίζουν και να ανταποκρίνονται σε ανθρώπινες συναισθηματικές εκφράσεις και εκφράσεις του προσώπου μέσω ενός τεχνητού νευρικού δικτύου/ συστήματος .

Θα επικεντρώνονται σε ανθρώπινα συναισθήματα που απαιτούν το ρομπότ να αντιδράσει ή να απαντήσει με ένα είδος συμπεριφοράς, όπως θυμό, οργή ή μοναξιά.. Είμαστε στην αυγή ενός κόσμου γεμάτου με αντικείμενα ευαίσθητα στα συναισθήματά μας; Αν αυτή η τεχνολογία βασίζεται κυρίως σε ένα λογισμικό, θα μπορούσαν τα επιτεύγματα σε αυτόν τον τομέα να χρησιμοποιηθούν και σε άλλα προϊόντα; Είναι συναρπαστικό και συγχρόνως τρομακτικό. Οι πιθανές εφαρμογές αυτής της τεχνολογίας θα είναι σίγουρα τεράστιες.

Οι τελευταίες τεχνολογικές εξελίξεις αποδεικνύουν ότι η πραγματικότητα καλπάζει πλέον με ρυθμούς πολύ ταχύτερους από αυτούς της επιστημονικής φαντασίας. Το μοναδικό που απομένει στους σκεπτόμενους ανθρώπους, πριν παρανοήσουν, είναι η ελπίδα ότι οι ευφυείς προγραμματιστές θα λειτουργούν με βάση το μέσο κοινό αίσθημα. Σε οποιαδήποτε άλλη περίπτωση, η ανθρωπότητα απλώς θα βρεθεί εξόριστη από τα ίδια της τα επιτεύγματα.

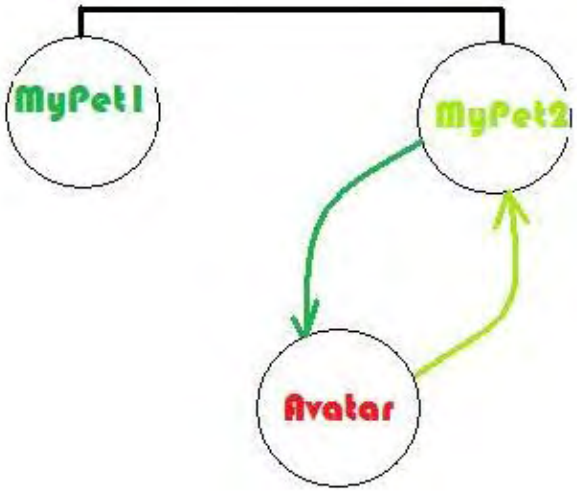
4.2 Συμπεράσματα / Αποτίμηση

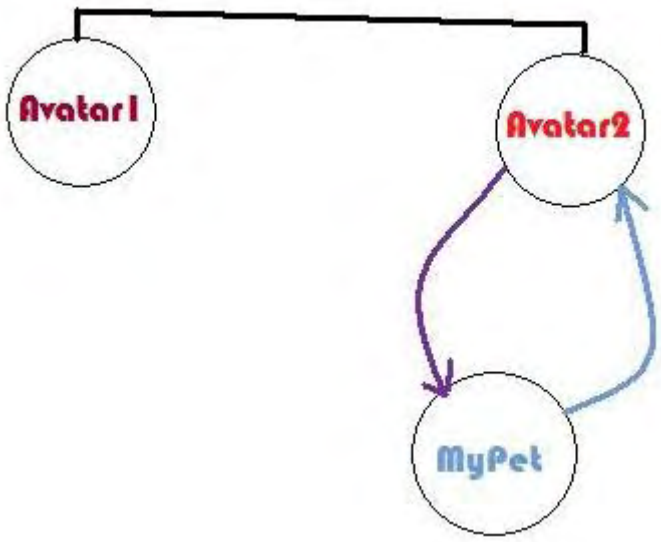
Με το πρόγραμμα MyPet v1.0 και v2.0 επιχείρησα να δώσω μια ανθρώπινη διάσταση στην έννοια του πράκτορα, δίνοντάς του ανθρώπινα συναισθήματα. Προσπάθησα να δω τον πράκτορα όχι απλά σαν μια οντότητα που εκτελεί ενέργειες ανεξάρτητα από εμάς (ή από έναν άλλο πράκτορα, όπως συμβαίνει στο MyPet v.2), αλλά σαν μια οντότητα η οποία έχει δικά της *συναισθήματα.* Τα συναισθήματα αυτά του Mypet (ευτυχία, πείνα, υγεία) αλλά και του Avatar (ευτυχία, πόσο απασχολημένος είναι) επηρεάζονται τόσο από την πάροδο του χρόνου όσο και από τις αλληλεπιδράσεις μεταξύ τους (και του MyPet μαζί μας).

Τα συναισθήματα είναι άμεσα συνδεδεμένα με τις ενέργειες του MyPet και του Avatar. Το μοντέλο αυτό των πρακτόρων, αντιγράφει ως ένα βαθμό τον τρόπο με τον οποίο λειτουργεί και ο άνθρωπος. Τα συναισθήματα είναι πολύ σημαντικά για τον άνθρωπο. Για παράδειγμα, εάν ένα άτομο καταδιώκεται από μία αρκούδα, νιώθει φόβο και

μαθαίνει από αυτή την εμπειρία να μην πλησιάζει αυτά τα ζώα. Τα ρομπότ έχουν ανάγκη από έναν παρόμοιο μηχανισμό. Εάν διαθέτεις κάτι το οποίο δεν έχει συναισθήματα, τότε αυτό δεν έχει ούτε στόχους αλλά ούτε και λόγο για να δράσει. Τα συναισθήματα είναι η ανταμοιβή ή η τιμωρία η οποία θα παρακινήσει το ρομπότ να πραγματοποιήσει τους στόχους του.

Στο MyPet v.2 ο Avatar δεν είναι εγγυημένο ότι θα αλληλεπιδράσει με το MyPet αυτό εξαρτάται από το αν είναι απασχολημένος. Επίσης ο Avatar είναι έτσι παραμετροποιημένος ώστε αν έχει κακή διάθεση δύναται να συναναστραφεί με άσχημο τρόπο προς το MyPet. Έτσι λόγω της τυχαιότητας που έχει η συμπεριφορά του Avatar, προσομοιώνει ως έναν βαθμό την συμπεριφορά του ανθρώπου.


MyPet1
MyPet2
Avatar


Avatar1
Avatar2
MyPet

5

Βιβλιογραφία

[1] http://el.wikipedia.org/wiki/Ρομπότ

[2] περιοδικό Focus τεύχος No 18, Αύγουστος 2001, σελ. 72-78

[3] Βασιλειάδου Χρυσούλα, Ρομπότ με συναισθήματα (ΤΕΙ ΣΕΡΡΩΝ τμήμα πληροφορικής και επικοινωνιών)

[4] Κώστας Δεληγιάννης, Ρομπότ με συναισθήματα Πρωτοποριακό πρόγραμμα από το Εργαστήριο Πολυμέσων του ΕΜΠ (Εφήμερίδα Καθημερινή), *http://news.kathimerini.gr/4dcgi/_w_articles_world_2_02/05/2010_399527*

[5] Σοφία Πάνου, Ρομπότ

http://sp-naturalsciences.blogspot.com/2009/07/2_12.html

[6] Εφημερίδα Telegraph, Ρομπότ με συναισθήματα (μετάφραση από Pathfinder) http://news.launcher.pathfinder.gr/periscopio/android-feeling.html

[7] http://el.wikipedia.org/wiki/Ταμαγκότσι

[8] Μαρία Αδαμίδου, Ρομπότ, προσωπικός διατροφολόγος... http://www.ethnos.gr/article.asp?catid=22733&subid=2&pubid=23740956

[9] Ομαδική τρέλα για το Farmville

http://www.espressonews.gr/default.asp?artID=1152745&catID=11&pid=79

[10] Brian Goetz, Introduction to Java threads (developer works), 26 Sep 2002

http://www.ibm.com/developerworks/java/tutorials/j-threads/j-threads-pdf.pdf

[11] http://users.cs.teilar.gr/gkakaron/java/Index.html

Java: Μια αντικειμενοστραφής γλώσσα προγραμματισμού Παρουσίαση της γλώσσας Java, από τον Κακαρόντζα Γιώργο.