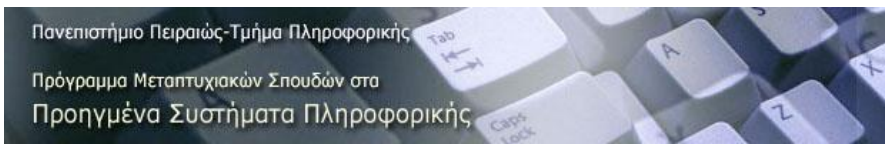

Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής»


Μεταπτυχιακή Διατριβή

| | |
|---|---|
| Τίτλος Διατριβής | **Adventurous Agents in REVE Virtual Environment** |
| Ονοματεπώνυμο Φοιτητή | **Αποστόλου Άρτεμις – Λουκρητία του Κωνσταντίνου** |
| Αριθμός Μητρώου | **ΜΠΣΠ/08026** |
| Κατεύθυνση | **Ευφυείς Τεχνολογίες Επικοινωνίας Ανθρώπου - Υπολογιστή** |
| Επιβλέπων | **Θεμιστοκλής Παναγιωτόπουλος, Καθηγητής** |




Ημερομηνία Παράδοσης **10/2011**

**Τριμελής Εξεταστική Επιτροπή**

| | | |
|---|---|---|
| Δημήτριος Δεσπότης<br>Καθηγητής | Θεμιστοκλής Παναγιωτόπουλος<br>Καθηγητής | Ευάγγελος Φούντας<br>Καθηγητής |

# Περιεχόμενα

*1. Περίληψη / Abstract ..... 5*

*2. Εισαγωγή – Σύντομη Περιγραφή Προβλήματος/Αντικειμένου ..... 6*

*3. Ευφυή Εικονικά Περιβάλλοντα και Ευφυής Πράκτορες ..... 7*

*4. Τεχνολογίες Υλοποίησης ..... 8*

**4.1 Vrml και Vrmlpad ..... 8**

**4.2 Η Αναπαράσταση REVE (Representation and Execution of Virtual Environments) .8**

**4.3 Η γλώσσα VERL ..... 8**

**4.4 REVE Worlds ..... 9**

**4.5 Reve Agents ..... 11**

4.5.1 SARA ..... 12

4.5.2 HouseKeeper ..... 13

*5. Αρχική υλοποίηση του εικονικού περιβάλλοντος και υλοποίηση σε VRML ..... 14*

**5.1 Αρχική υλοποίηση του εικονικού περιβάλλοντος ..... 14**

**5.2 Γεωμετρικά αντικείμενα ..... 14**

**5.3 Εμφάνιση Αντικειμένων – Κόμβος Appearance ..... 15**

**5.4 Lighting Nodes ..... 16**

**5.5 Ομαδοποίηση κόμβων ..... 18**

**5.6 Υλοποίηση γεγονότων βασισμένων σε timers και σε ενέργειες του ήρωα ..... 19**

**5.7 Ήχοι ..... 21**

**5.8 NavigationInfo ..... 21**

**5.9 Ο κόμβος ViewPoint ..... 22**

**5.10 Ο κόμβος Background ..... 22**

**5.11 Ο κόμβος Fog ..... 22**

*6. Αναπαράσταση του κόσμου σε VERL ..... 24*

**6.1 Virtual Model Definition ..... 24**

6.1.1 Ορισμός ..... 24

6.1.2 Προβλήματα Υπολογισμού Bounding Boxes ..... 26

**6.2 Access Model Definition ..... 40**

6.2.1 Ορισμός ..... 40

6.2.2 Προβλήματα Εφαρμογής Functions στον κόσμο μας ..... 45

**6.3 Semantic Model Definition ..... 50**

6.3.1 Ορισμός ..... 50

*7. Εικονικοί Πράκτορες ..... 53*

**7.1 Walkthrough ..... 53**

**7.2 Ο Πράκτορας HouseKeeper ..... 62**

7.2.1 Συναρτήσεις για items τα οποία πραγματοποιούν κατά την κίνηση τους translation ..... 66
7.2.2 Συναρτήσεις για items τα οποία πραγματοποιούν κατά την κίνηση τους rotation ..... 70
7.2.3 Συναρτήσεις για κόμβους items τα οποία πραγματοποιούν κατά την κίνηση τους translation..... 74
7.2.4 Συναρτήσεις για κόμβους items τα οποία πραγματοποιούν κατά την κίνηση τους rotation 79

**7.3 SARA Agent..... 83**
7.3.1 Συναρτήσεις για items τα οποία πραγματοποιούν κατά την κίνηση τους translation ..... 96
7.3.2 Συναρτήσεις για items τα οποία πραγματοποιούν κατά την κίνηση τους rotation ..... 97
7.3.3 Συναρτήσεις για κόμβους items τα οποία πραγματοποιούν κατά την κίνηση τους translation..... 99
7.3.4 Συναρτήσεις για κόμβους items τα οποία πραγματοποιούν κατά την κίνηση τους rotation 101
7.3.5 Συναρτήσεις για την σύγκριση συμβόλων ..... 103

***8. Συμπεράσματα – Περίληψη ..... 105***

***9. Βιβλιογραφία ..... 108***

# 1. Περίληψη / Abstract

Τα ευφυή εικονικά περιβάλλοντα είναι ένα νέο σχετικά επιστημονικό πεδίο, το οποίο συνδυάζει αρχές ανάπτυξης τρισδιάστατων γραφικών, τεχνητής νοημοσύνης και ευφυών πρακτόρων, αντιμετωπίζοντας ένα πλήθος προβλημάτων αναπαράστασης και μοντελοποίησης. Το αποτέλεσμα είναι ένα πλήθος ρεαλιστικών και ελκυστικών εφαρμογών για διάφορους τομείς.

Στην παρούσα μεταπτυχιακή διατριβή παρουσιάζεται η αναπαράσταση REVE για την ανάπτυξη εικονικών περιβαλλόντων, η γλώσσα VERL για ανάπτυξη εικονικών κόσμων με βάση την αναπαράσταση REVE, η πλατφόρμα REVE Worlds για την αναπαράσταση αυτών των εικονικών κόσμων και αναπτύσσονται ευφυής εικονικοί πράκτορες ειδικά για κόσμους αναπτυγμένους με την αναπαράσταση REVE. Σκοπός είναι η ανάπτυξη ενός πλήρους και λειτουργικού ευφυούς εικονικού περιβάλλοντος με σκοπό την εξερεύνηση των δυνατοτήτων των παραπάνω τεχνολογιών, η αντιμετώπιση προβλημάτων και η πρόταση λύσεων και βελτιώσεων.

Intelligent virtual environments are a new scientific field which combines aspects of 3d graphics design, artificial intelligence and intelligent virtual agents, facing a wide range of difficulties considering representation and modeling. As a result there's been a wide variety of realistic and attractive applications covering a range of areas.

This postgraduate thesis presents the REVE representation for developing virtual environments, the language VERL for developing virtual environments based on REVE representation, REVE Worlds a platform for the representation of those virtual environments and artificial intelligent agents are being developed especially for worlds developed with REVE representation. The aim is the development of a comprehensive and operating intelligent virtual environment exploring the potentials of the above technologies, troubleshooting and proposing solutions and improvements.

# 2. Εισαγωγή – Σύντομη Περιγραφή Προβλήματος/Αντικειμένου

Η παρούσα μεταπτυχιακή διατριβή πραγματεύεται την ανάπτυξη ενός παιχνιδιού περιπέτειας (Adventure Game) με ευφυείς εικονικούς πράκτορες κατά την περιπλάνηση τους σε έναν εικονικό κόσμο.

Η αρχική σύλληψη της ιδέας έγινε στα πλαίσια του μαθήματος «Ευφυή Εικονικά Περιβάλλοντα» του μεταπτυχιακού προγράμματος σπουδών «Προηγμένα Συστήματα Πληροφορικής» του Πανεπιστημίου Πειραιώς. Για τις ανάγκες του μαθήματος δημιουργήθηκε ένα εικονικό περιβάλλον στο οποίο ο χρήστης περιπλανιέται σε ένα κάστρο με στόχο να λύσει κάποιους γρίφους και να μπορέσει να βρει την έξοδο. Το σενάριο αυτό αποτελούνταν από δύο σκέλη, από τη μία όπως προαναφέρθηκε ο ήρωας βρίσκεται εγκλωβισμένος σε ένα κάστρο χωρίς να γνωρίζει πως και γιατί βρέθηκε εκεί προσπαθεί να βρει μια έξοδο ενώ από την άλλη υπάρχει ένας πράκτορας «θεός» του κόσμου ο οποίος προσπαθεί να εμποδίσει τον ήρωα τον ήρωα από το να φτάσει στον σκοπό του. Το απλό αυτό σενάριο επέτρεψε να υλοποιηθούν διαφορετικές πτυχές της ύλης του μαθήματος Ευφυή Εικονικά Περιβάλλοντα. Από την μια μεριά, η όλη προσπάθεια του ήρωα να διαφύγει δημιουργεί ένα παιχνίδι (με γραφικά, μικρά animations, χειρισμό του ήρωα από τον χρήστη κλπ) τα οποία υλοποιήθηκαν σε vrml και αντιστοιχούν στα πρώτα επίπεδα της πυραμίδας της εικονικής πραγματικότητας (γεωμετρικό, κινηματικό και συμπεριφορικό). Από την άλλη πλευρά η υλοποίηση ενός αόρατου εχθρού ο οποίος δρα από μόνος του αντιστοιχεί στο κομμάτι του μαθήματος που ασχολείται με την τεχνητή νοημοσύνη.

Η αρχική σκέψη της εργασίας ήταν αρκετά φιλόδοξη: η αναπαράσταση του χώρου και η συμπεριφορά του κακού θα υλοποιούνταν στην προγραμματιστική γλώσσα java, θα γίνονταν η σύνδεση της java με την vrml και ο τελικός στόχος ήταν οι δύο εφαρμογές να τρέχουν ταυτόχρονα, έτσι ώστε ότι ενέργεια πραγματοποιούσε ο χρήστης η οποία είχε επιρροή στον κόσμο του παιχνιδιού να απεικονιζόταν κατευθείαν στην java, και να άλλαζε την συμπεριφορά του κακού. Το σχέδιο αυτό δεν έφτασε στα τελικά του στάδια λόγω έλλειψης χρόνου στα πλαίσια μιας εργασίας μαθήματος εξαμήνου αλλά κυρίως λόγω της πολυπλοκότητας της υλοποίησης. Τα δύο ξεχωριστά κομμάτια (vrml-απεικόνιση παιχνιδιού και java-απεικόνιση συμπεριφοράς κακού) υλοποιήθηκαν ξεχωριστά σε έναν αρκετά ικανοποιητικό βαθμό, χωρίς επιτευχθεί η σύνθεση των δύο αυτών κομματιών.

Τα προβλήματα και οι δυσκολίες που ανέκυψαν στην προσπάθεια της παραπάνω υλοποίησης ξεπεράστηκαν στην παρούσα διατριβή με τη χρήση του συστήματος REVE Worlds, μιας ιδιαίτερα φιλικής πλατφόρμας η οποία χρησιμοποιεί την αναπαράσταση REVE (Representation and Execution of Virtual Environments) και διευκολύνει την υλοποίηση εικονικών κόσμων και την ανάπτυξη ευφυών πρακτόρων. Έτσι η αρχική υλοποίηση άλλαξε ελαφρώς ο χρήστης αντικαταστάθηκε από έναν ευφυή εικονικό πράκτορα, ενώ ο πράκτορας «θεός» του κόσμου ενημερώνεται για οτιδήποτε συμβαίνει σε αυτόν και ανάλογα φροντίζει για τις διάφορες ενέργειες που προκαλούνται στον κόσμο.

# 3. Ευφυή Εικονικά Περιβάλλοντα και Ευφυής Πράκτορες

Η ραγδαία ανάπτυξη της επιστήμης των υπολογιστών τα τελευταία χρόνια επέτρεψε την ανάπτυξη όλο και πιο ισχυρών από άποψη υλικού μηχανημάτων και την διάδοση τους στο ευρύ κοινό. Η ανάπτυξη αυτή είχε ως αποτέλεσμα την ανάπτυξη σύνθετων προγραμμάτων λογισμικού με τομείς ανάπτυξης σε πλήθος εφαρμογών. Παράλληλα με την ανάπτυξη της επιστήμης της τεχνητής νοημοσύνης και την ανάπτυξη των γραφικών και κατ' επέκταση των τρισδιάστατων γραφικών δόθηκε η δυνατότητα ανάπτυξης σύνθετων συστημάτων απεικόνισης τρισδιάστατων κόσμων εμπλουτισμένα με τεχνικές από τον χώρο της τεχνητής νοημοσύνης.

Ως εικονικά περιβάλλοντα (virtual environments) ορίζονται συνθετικοί κόσμοι οι οποίοι διακρίνονται από ένα σύνολο κανόνων και επιτρέπουν στον χρήστη την πλοήγηση και αλληλεπίδραση με αυτά. Οι τομείς της εφαρμογής τους καλύπτουν ένα ευρύ φάσμα καθώς βρίσκουν εφαρμογή στη διασκέδαση και τις τέχνες, στο χώρο της εκπαίδευσης και της προσομοίωσης. Οι κόσμοι οι οποίοι δημιουργούνται είναι ιδιαίτερα ρεαλιστικοί και προσφέρουν έτσι εξαιρετικά ενδιαφέρουσες εφαρμογές.

Ένας εικονικός πράκτορας (virtual agent) είναι ένα σύστημα το οποίο διέπεται από αυτονομία στη δράση του σε ένα εικονικό περιβάλλον. Ένας ευφυής εικονικός πράκτορας (intelligent virtual agent) έχει ως πρόσθετα χαρακτηριστικά την αυτονομία, την προσαρμοστικότητα και την κοινωνικότητα. Χρησιμοποιεί κάποιους αισθητήρες (sensors) για συλλογή δεδομένων και δρα στο περιβάλλον με τη βοήθεια μηχανισμών δράσης (effectors).

Όταν τα εικονικά περιβάλλοντα διέπονται από αρχές της τεχνητής νοημοσύνης και των ευφυών πρακτόρων τότε έχουμε την δημιουργία ευφυών εικονικών περιβαλλόντων, ενός νέου σχετικά επιστημονικού πεδίου.

# 4. Τεχνολογίες Υλοποίησης

## 4.1 Vrml και Vrmlpad

Η VRML (Virtual Reality Modeling Language) είναι μια γλώσσα ειδικά σχεδιασμένη για την δημιουργία και αναπαράσταση τρισδιάστατων διανυσματικών γραφικών. Είναι ουσιαστικά η τρισδιάστατη εκδοχή της HTML και επιτρέπει την δημιουργία κόσμων αλλά και αλληλεπιδράσεων (interactions) με τον χρήστη. Έτσι ο χρήστης μπορεί να παρατηρήσει, να πλοηγηθεί, να περιστρέψει ή να αλληλεπιδράσει με την τρισδιάστατη σκηνή χρησιμοποιώντας τα ειδικά κουμπιά που προσφέρει το περιβάλλον πλοήγησης. Για τις ανάγκες της παρούσας μεταπτυχιακής διατριβής χρησιμοποιήθηκε η έκδοση vrml 2.0. Επίσης βοηθητικό εργαλείο αποτέλεσε το VrmlPad 3.0, ένας editor ειδικά σχεδιασμένος για τη δημιουργία και διαχείριση εικονικών περιβαλλόντων δημιουργημένων σε VRML.

## 4.2 Η Αναπαράσταση REVE (Representation and Execution of Virtual Environments)

Η αναπαράσταση REVE αποτελεί μία αφηρημένη αναπαράσταση για εικονικά περιβάλλοντα η οποία στοχεύει στην τυποποιημένη ανάπτυξη εικονικών περιβαλλόντων ενώ προσφέρει ευελιξία, επεκτασιμότητα καθώς και ρεαλιστικό σχεδιασμό.

Ένα εικονικό περιβάλλον ορίζεται από ένα σύνολο εικονικών αντικειμένων τα οποία περιέχουν το σύνολο της διαθέσιμης πληροφορίας του κόσμου. Γίνεται λοιπόν σαφές ότι οτιδήποτε θα θέλαμε να ενσωματώσουμε στα πλαίσια του κόσμου μας θα πρέπει να αντιστοιχεί σε κάποιο αντικείμενο. Για αυτό το λόγο κάθε αντικείμενο αναπαριστάται από μια δομή δεδομένων η οποία καλείται item. Στον πραγματικό κόσμο τα φυσικά αντικείμενα τα οποία αντιλαμβάνεται ο άνθρωπος διέπονται από ένα σύνολο χαρακτηριστικών τόσο φυσικών όσο και λειτουργικών. Έτσι και στην δομή item θα πρέπει να περιέχονται όλα τα χαρακτηριστικά του αντικειμένου τόσο από φυσικής όσο και από λειτουργικής άποψης. Πιο συγκεκριμένα θα πρέπει να περιέχονται πληροφορίες για το μέγεθος, το προσανατολισμό, την τοποθεσία στην οποία βρίσκεται το item κ.τ.λ. (φυσική αναπαράσταση) αλλά και πληροφορίες σχετικά με την λειτουργικότητα του όπως είναι η δυνατότητα μετακίνησης, περιστροφής και γενικότερα κάθε αλλαγής ως προς την φυσική αναπαράσταση του item (αναπαράσταση λειτουργικότητας). Επιπρόσθετα θα οι πληροφορίες θα πρέπει να είναι διαθέσιμες ως προς κάθε δέκτη με ένα κατάλληλο επίπεδο αφαίρεσης έτσι ώστε ο κάθε δέκτης να μπορεί να χρησιμοποιήσει την πληροφορία με τον τρόπο που θέλει καθώς και στο εύρος που θέλει (σημασιολογική αναπαράσταση).

Στην αναπαράσταση REVE προκειμένου να καλυφθούν οι παραπάνω κατηγορίες αναπαράστασης εισάγονται τρία item aspects ως επιμέρους δομές ενός item, τρεις δηλαδή «απόψεις» αναπαράστασης της πληροφορίας οι οποίες είναι οι ακόλουθες:

- Physical aspect:

  Η οποία αντιστοιχεί στην φυσική αναπαράσταση και σχετίζεται με τα «φυσικά» χαρακτηριστικά του εικονικού item (μέγεθος, σχήμα, κ.τ.λ.)

- Semantic aspect:

  Η οποία αντιστοιχεί στην σημασιολογική αναπαράσταση και σχετίζεται με την πληροφορία την οποία μπορούν να δεχθούν οι δέκτες (εδώ εικονικοί πράκτορες)

- Virtual aspect:

  Η οποία αντιστοιχεί στην λειτουργική αναπαράσταση και σχετίζεται με την λειτουργικότητα του item.

## 4.3 Η γλώσσα VERL

Ένα πολύ σημαντικό κομμάτι της αναπαράστασης REVE είναι η γλώσσα VERL (Virtual Environment Representation Language) μια απλή γλώσσα βασισμένη στην μεταγλώσσα XML η οποία είναι ειδικά

σχεδιασμένη για την αναπαράσταση εικονικών περιβαλλόντων και ακολουθεί τις αρχές τις αναπαράστασης REVE.

Στην γλώσσα VERL ορίζεται ο εικονικός κόσμος (world) ο οποίος αποτελείται από items τα οποία ανήκουν σε συγκεκριμένα itemGroups. Σε κάθε ένα από αυτά τα itemGroups ή τα items μπορούν να αποδοθούν συγκεκριμένα χαρακτηριστικά όπως π.χ. class:real/unreal, behaviors: true/false κ.τ.λ. Επιπρόσθετα αναπαριστούνται τα τρία παραπάνω item aspects για κάθε item. Έτσι μια αντιπροσωπευτική αναπαράσταση ενός αντικειμένου είναι η ακόλουθη:

```
<itemGroup name="doors" location="models/castle/doors/doors.wrl">

        <item name="chest_room4">
                <virtualModel source="chest">
                        <transform  translation="4.7, 0, -11.2" rotation="0 1 0  1.57" />
                </virtualModel>

                <accessModel>
                        <accessPoint name="chest_r4_open_access_p" node="top">
                                <function name="rotate"
                                  class="reve.scene.item.access.function.X3DFieldFunctionL1"
                                   args="rotation"
                                />
                        </accessPoint>
                </accessModel>

                <semanticModel>
                        <symbol name="chestr4_open_symbol">
                                <argument
                                  class="reve.scene.item.semantic.argument.X3DConnectorL1"
                                  args="top, translation"
                                />
                        </symbol>
                </semanticModel>

        </item>

</itemGroup>
```

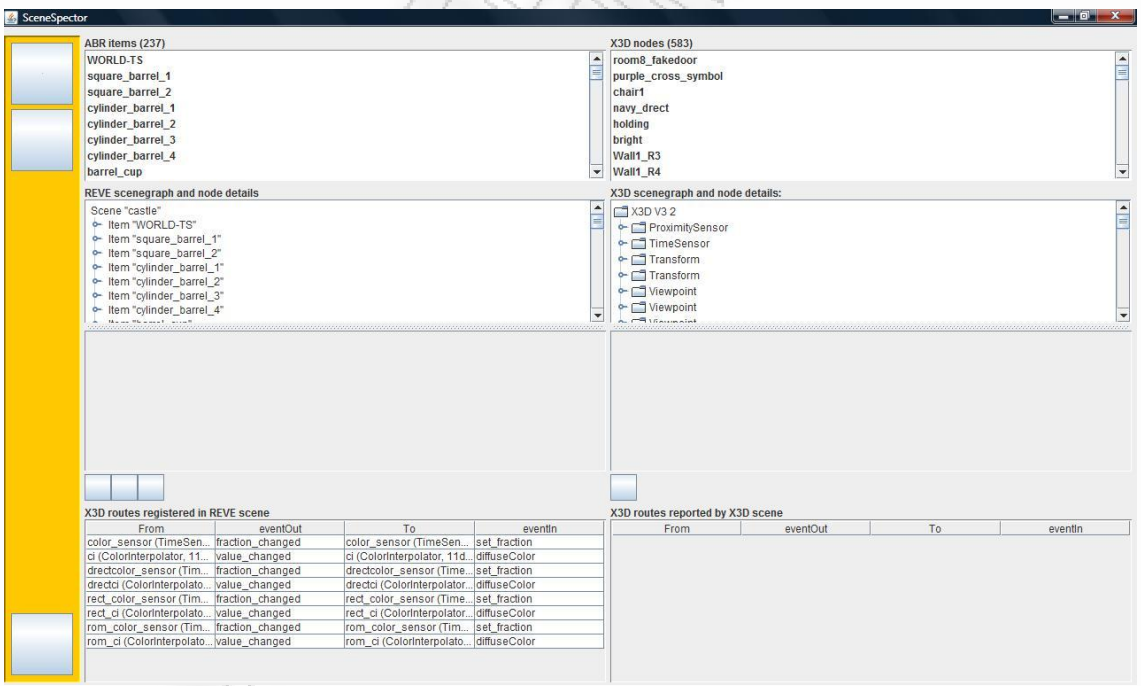
Περισσότερα σχετικά με την γλώσσα VERL θα αναφερθούν παρακάτω σε αντίστοιχο κεφάλαιο.

## 4.4 REVE Worlds

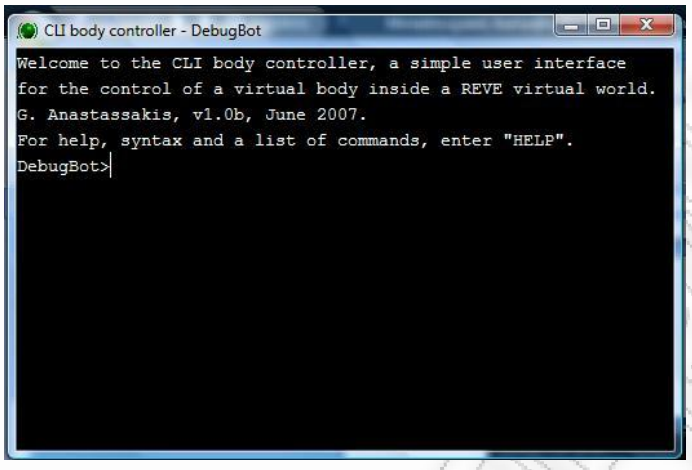
Το REVE Worlds είναι μία ολοκληρωμένη πλατφόρμα υλοποιημένη και αυτή σύμφωνα με την αναπαράσταση REVE η οποία επιτρέπει την ρεαλιστική απεικόνιση εικονικών κόσμων μεταφορτώνοντας αρχεία VERL και υποστηρίζει την ανάπτυξη και παρακολούθηση εικονικών πρακτόρων.


**REVE Worlds**

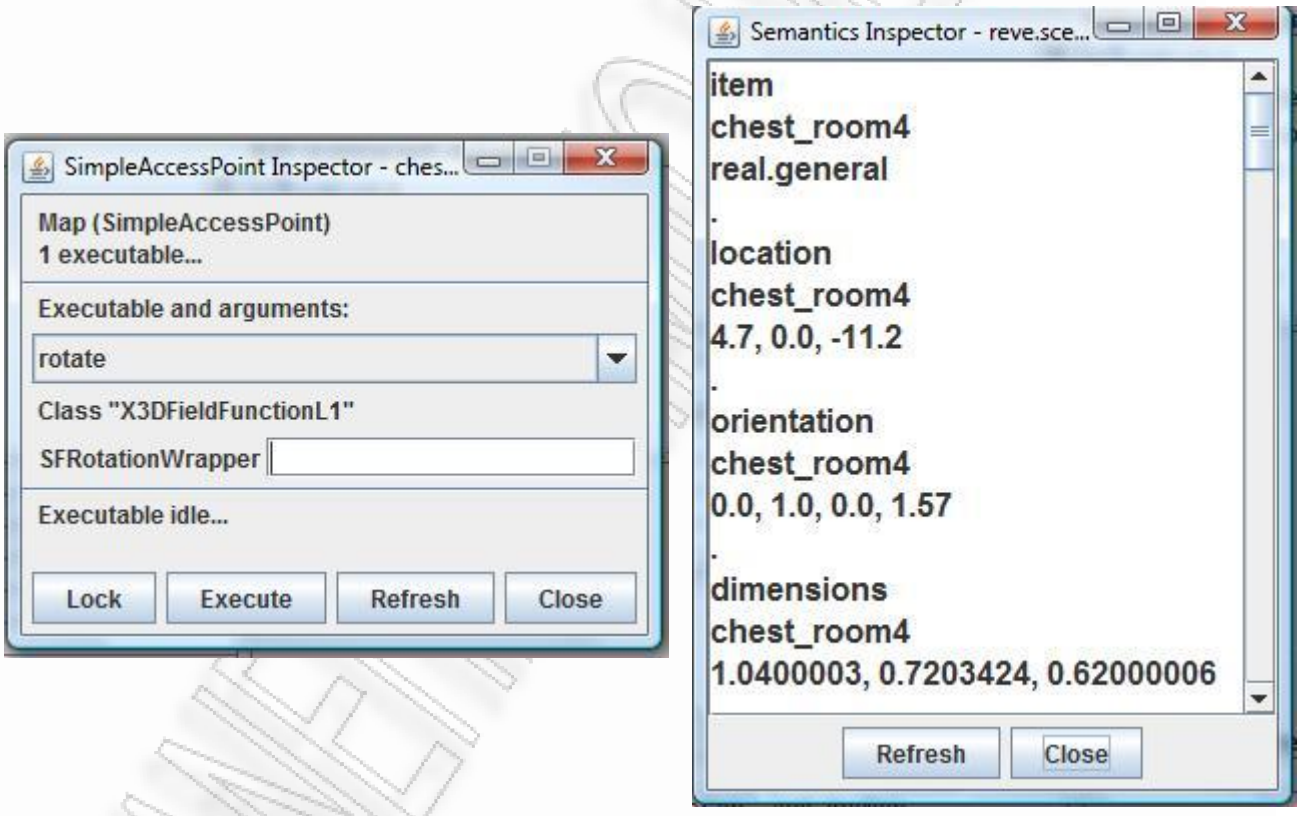
Το REVE Worlds προσφέρει εύκολο και λειτουργικό σύστημα πλοήγησης, πληροφορίες για το σύνολο των items που αποτελούν τον κόσμο, μέσω του Scene Inspector, παρέχοντας και επιπλέον πληροφορίες σχετικά με αυτά όπως π.χ. τον υπολογισμό του μεγέθους των bounding boxes, πλήρη έλεγχο εικονικών σωμάτων μέσω του CLI body Controller, δυνατότητα χειροκίνητου ελέγχου της λειτουργικότητας των items μέσω του Accesspoint Inspector και τέλος πληροφορίες για το σύνολο των σημασιολογικών χαρακτηριστικών των items μέσω του Semantic Inspector.




**Scene Inpector**



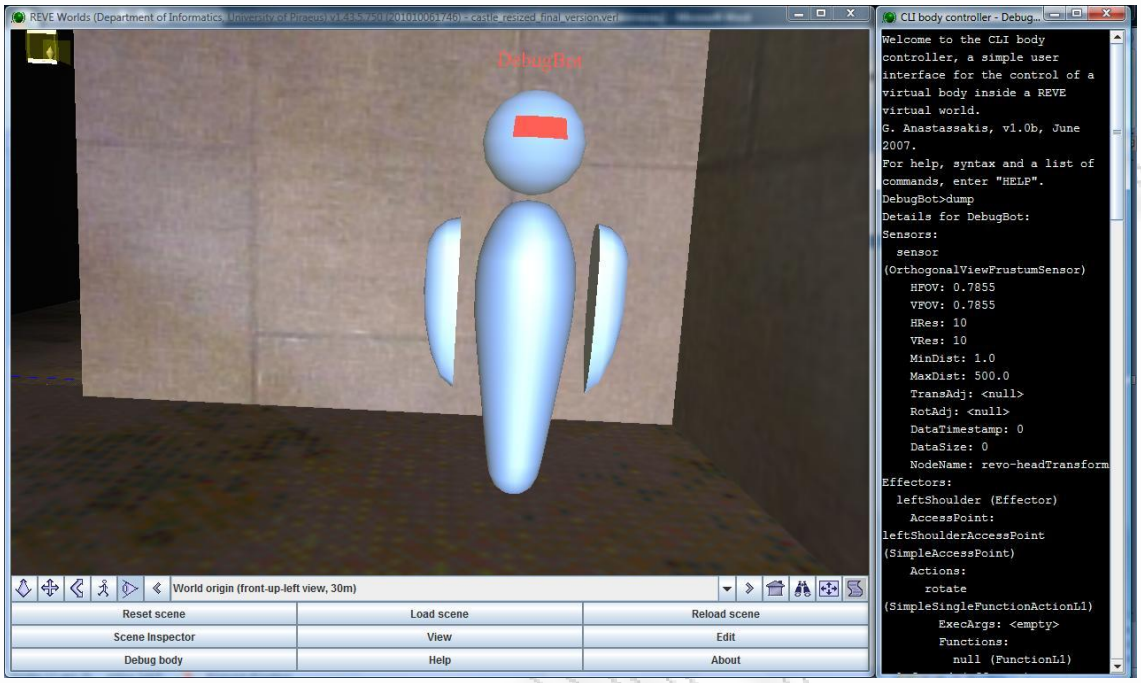

**CLI body controller**




**AccessPoint Inspector & Semantics Inspector**

## 4.5 Reve Agents

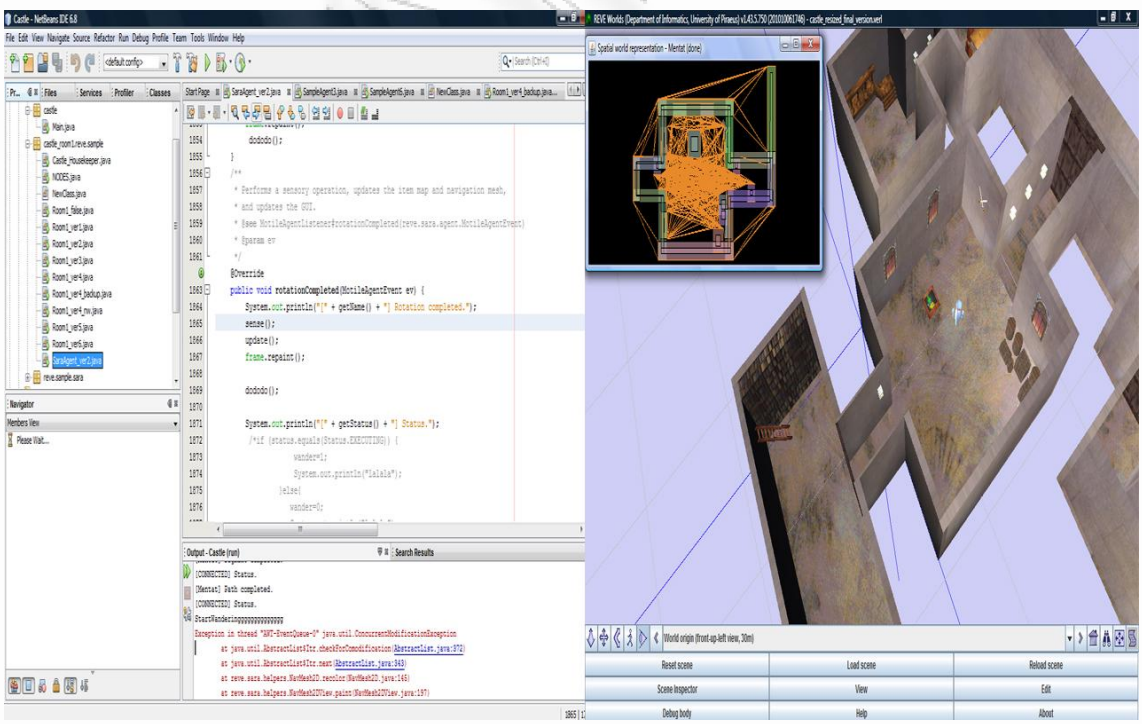
Όπως προαναφέρθηκε τόσο η αναπαράσταση REVE υποστηρίζει την παρουσία εικονικών σωμάτων στον κόσμο, όσο και το REVE Worlds διευκολύνει την παρουσία τους και τον έλεγχο τους στον κόσμο. Μέσω του REVE Worlds λοιπόν είναι δυνατή η φόρτωση στον κόσμο πρακτόρων τα οποία ονομάζονται DebugBots και τα οποία ο χρήστης μπορεί να ελέγξει μέσω του CLI body Controller όπως έχει αναφερθεί και παραπάνω. Οι πράκτορες αυτοί μπορούν να κινηθούν στον κόσμο, να κάνουν sense σε αυτόν και να αλληλεπιδράσουν με αυτόν χρησιμοποιώντας τους δικούς τους effectors σε σημεία του κόσμου τα οποία περιέχουν λειτουργικότητα προκειμένου να προκαλέσουν αλλαγές σε αυτόν.




**DebugBot**

### 4.5.1 SARA

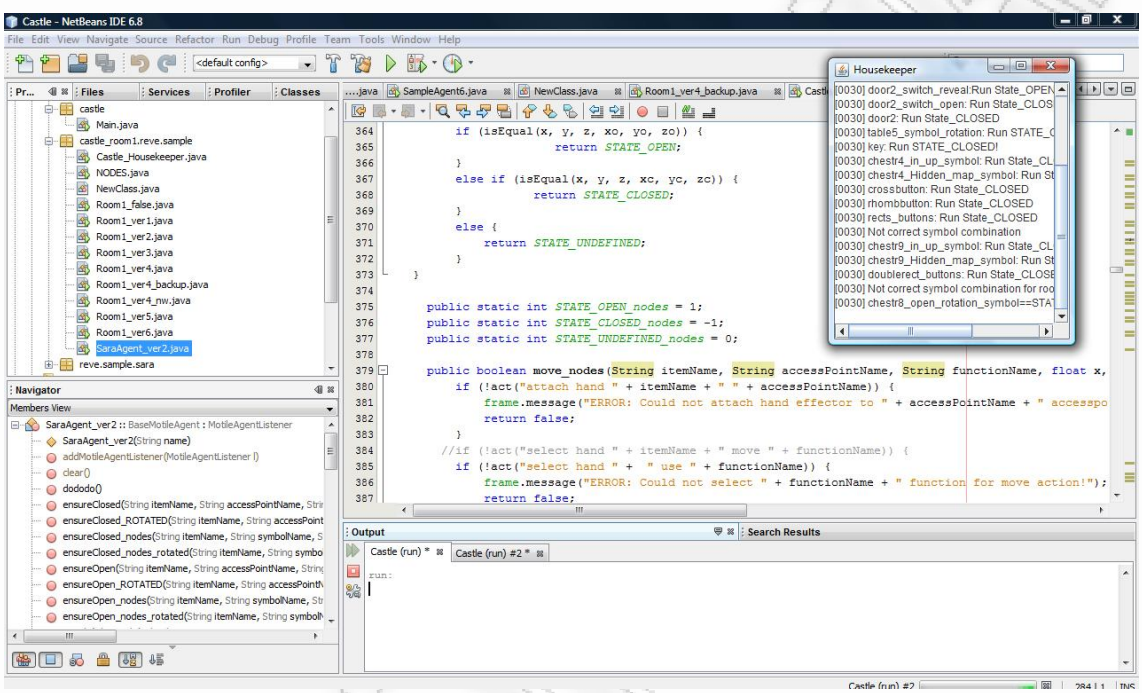
Η SARA (Simple API for REVE Agents) είναι μια βιβλιοθήκη υλοποιημένη σε JAVA η οποία συνδυάζει την λειτουργικότητα των απλών εικονικών σωμάτων (DebugBots) με πολύπλοκες διαδικασίες pathfinding, wandering, χωρικής αναπαράσταση, ελέγχου της βάσης γνώσης και εξαγωγής συμπερασμάτων από αυτή καθώς και άλλων πολύπλοκων διαδικασιών. Μπορούμε να πούμε ότι ένας SARA agents είναι ένας DebugBot με αυξημένες ικανότητες.




**SARA**

### 4.5.2 HouseKeeper

Ο HouseKeeper είναι μια βιβλιοθήκη υλοποιημένη σε JAVA η οποία υποστηρίζει την ύπαρξη ειδικών πρακτόρων, άυλων οι οποίοι είναι υπεύθυνοι για το σύνολο του κόσμου, έχουν την γενική εποπτεία του καθώς και διευρυμένη γνώση για αυτόν και μπορούν να προκαλέσουν ενέργειες είτε από μόνοι τους είτε ως συνέπεια μιας αλληλεπίδρασης στον κόσμο. Οι πράκτορες HouseKeeper έχουν δηλαδή το ρόλο ενός εικονικού θεού στον εικονικό μας κόσμο, όπου τίποτα δεν μπορεί να εκτελεστεί αν δεν το «εγκρίνουν» οι ίδιοι.




**HouseKeeper Agent**

# 5. Αρχική υλοποίηση του εικονικού περιβάλλοντος και υλοποίηση σε VRML

## 5.1 Αρχική υλοποίηση του εικονικού περιβάλλοντος

Η αρχική υλοποίηση του γραφικού εικονικού περιβάλλοντος έγινε χρησιμοποιώντας VRML (Virtual Reality Modeling Language) μια γλώσσα ειδικά σχεδιασμένη για την δημιουργία και αναπαράσταση τρισδιάστατων διανυσματικών γραφικών. Επίσης βοηθητικό εργαλείο αποτέλεσε το VrmlPad 3.0, ένας editor ειδικά σχεδιασμένος για τη δημιουργία και διαχείριση εικονικών περιβαλλόντων δημιουργημένων σε VRML. Παρακάτω αναφέρονται οι δυνατότητες τις VRML οι οποίες χρησιμοποιήθηκαν για τη δημιουργία του αρχικού κόσμου. Στόχος δεν είναι η ανάλυση όλων των δυνατοτήτων της VRML, αλλά η παρουσίαση των τεχνικών οι οποίες βοήθησαν στην υλοποίηση της παρούσας μεταπτυχιακής διατριβής.

## 5.2 Γεωμετρικά αντικείμενα

Όλα τα αντικείμενα στον κόσμο είναι ένας συνδυασμός των ακόλουθων βασικών γεωμετρικών σχημάτων της VRML:

- Box
  - Ο κόμβος Box καθορίζει ένα ορθογώνιο παραλληλεπίπεδο που επιτρέπει τον ορισμό του πλάτους, του ύψους, και του βάθους του ορθογωνίου.

```
geometry Box {
                size  20 20 1.4
        }
```

- Cone
  - Ο κόμβος Cone, καθορίζει έναν κώνο στον οποίο μπορούμε να καθορίσουμε το ύψος, την ακτίνα, καθώς και το ποιο μέρος του θα είναι ορατό (πλευρά ή βάση).

```
geometry Cone {
                bottomRadius 0.3
                height 0.5
                side TRUE
                bottom TRUE
        }
```

- Cylinder
  - Ο κόμβος Cylinder καθορίζει έναν κύλινδρο στον οποίο μπορούμε να καθορίσουμε το ύψος, την ακτίνα, καθώς και το ποιο μέρος του θα είναι ορατό (πλευρά ή βάση).

```
geometry Cylinder {
                radius 1.5
                height 0.5
                side TRUE
                bottom TRUE
                top TRUE
        }
```

- Sphere
  - Ο κόμβος Sphere καθορίζει μία σφαίρα της οποίας μπορούμε να καθορίσουμε την ακτίνα.

```
geometry Sphere {
                radius 0.012
        }
```

## 5.3 Εμφάνιση Αντικειμένων – Κόμβος Appearance

Ο κόμβος appearance καθορίζει την εμφάνιση των γεωμετρικών αντικειμένων. Περιλαμβάνει τα ακόλουθα πεδία:

- Material
  - Το πεδίο Material περιέχει με τη σειρά του ένα κόμβο material στον οποίο καθορίζεται το χρώμα του αντίστοιχου γεωμετρικού αντικειμένου, τον τρόπο με τον οποίο απεικονίζεται το φώς σε αυτό καθώς και για το εάν το αντικείμενο φαίνεται διαφανές ή όχι. Εάν το πεδίο έχει τιμή NULL, τότε όλα τα φώτα τα οποία έχουν οριστεί στον κόσμο δεν έχουν καμία επίδραση στο αντικείμενο. Περιέχει έξι πεδία:
    - diffuseColor: Καθορίζει το χρώμα της γεωμετρίας
    - emissiveColor: Καθορίζει την λάμψη των αντικειμένων
    - ambientIntensity: Καθορίζει το ποσοστό του φωτός που αντανακλάται στο αντικείμενο
    - specularColor: Καθορίζει το χρώμα των φωτεινών σημείων των αντικειμένων
    - shininess: Καθορίζει πόσο έντονη είναι η λάμψη στα φωτεινά σημεία των αντικειμένων
    - transparency: Καθορίζει το πόσο διάφανα θα είναι τα αντικείμενα

```
appearance Appearance {
    material Material {
                diffuseColor .35 .1 .1
                specularColor .37 .06 .06
                emissiveColor .1 .03 .03
                ambientIntensity .0233
                shininess .06
                transparency 0
        }
    }
```

- Texture
  - Το πεδίο texture περιέχει ένα κόμβο texture ο οποίος περιέχει μια ImageTexture. Σε περίπτωση όπου το πεδίο έχει τιμή NULL, τότε η εμφάνιση του αντίστοιχου γεωμετρικού αντικειμένου καθορίζεται μόνο από το χρώμα το οποίο έχουμε ορίσει σε αυτό.

```
appearance Appearance {
        texture   ImageTexture {
                url "wood1.jpg"
        }
    }
```

- Texture Transform
  - Το πεδίο TextureTransform καθορίζει τον τρόπο με τον οποίο εφαρμόζεται η υφή στο αντίστοιχο γεωμετρικό αντικείμενο. Καθορίζει απλούς γεωμετρικούς σχηματισμούς όπως αλλαγή κλίμακας, περιστροφή και μεταφορά στις συντεταγμένες της υφής

```
TextureTransform {
        scale 1 1
        rotation 0
        center 0 0
        translation 0 0
}
```

## 5.4 Lighting Nodes

Βασική πηγή φωτός στον κόσμο μας είναι το headlight, ένα φως το οποίο δημιουργείται από τον browser και προσαρμόζεται στην τρέχουσα οπτική γωνία (viewpoint) του ήρωα. Το headlight δείχνει πάντα προς το μέρος στο οποίο κοιτάει ο ήρωας και είναι ουσιαστικά ένα φως «κολλημένο» στο κεφάλι του. Το headlight μπορεί να είναι είτε ενεργοποιημένο είτε όχι. Στον δικό μας κόσμο είναι πάντα ενεργοποιημένο. Η συμπεριφορά του καθορίζεται στον κόμβο NavigationInfo ο οποίος θα αναλυθεί παρακάτω.

Επιπρόσθετα στη vrml υπάρχουν τρία ακόμα είδη φωτός:

- Directional Light

  Οι ακτίνες είναι παράλληλες. Το φώς δεν έχει καθορισμένες συντεταγμένες παρά μόνο κατεύθυνση. Λειτουργεί σαν το φως να είναι πολύ μακριά από τον κόσμο μας. Περιέχει τα ακόλουθα πεδία:
  - on: Καθορίζει εάν το φώς είναι ενεργό
  - intensity: Καθορίζει πόσο έντονο είναι το φώς και παίρνει τιμές οπό 0 έως 1
  - ambientIntensity: Καθορίζει το ποσοστό συμμετοχής στο συνολικό φωτισμό και παίρνει τιμές από 0 έως 1
  - color: είναι ένα RGB πεδίο που καθορίζει το χρώμα του φωτός
  - direction: Καθορίζει την κατεύθυνση του φωτός με ένα 3D διάνυσμα

```
DirectionalLight {
        on TRUE
        intensity 1
        ambientIntensity 1
        color 0.53 0.15 0.34
        direction 0 1 0
}
```

- Point Light

  Αυτό το είδος φωτός φωτίζει τα πάντα τα οποία βρίσκονται γύρω του και οι ακτίνες κατευθύνονται προς όλες τις κατευθύνσεις. Είναι μία πηγή φωτός σε ένα συγκεκριμένη θέση στον κόσμο μας. Φωτίζει όλους τους κόμβους ανεξάρτητα από τη θέση τους στο αρχείο. Για αυτό καθορίζουμε τη μέγιστη απόσταση στην οποία μπορεί να έχει επίδραση το φώς. Περιέχει τα ακόλουθα πεδία:
  - on: Καθορίζει εάν το φώς είναι ενεργό
  - intensity: Καθορίζει πόσο έντονο είναι το φώς και παίρνει τιμές οπό 0 έως 1
  - ambientIntensity: Καθορίζει το ποσοστό συμμετοχής στο συνολικό φωτισμό και παίρνει τιμές από 0 έως 1
  - color: είναι ένα RGB πεδίο που καθορίζει το χρώμα του φωτός
  - location: Καθορίζει την θέση του φωτός με ένα 3D διάνυσμα

- attenuation: Είναι ένα 3D διάνυσμα το οποίο καθορίζει τον τρόπο με τον οποίο το φώς χάνει την ένταση του καθώς απομακρυνόμαστε από την πηγή.
- radius: Καθορίζει την μέγιστη ακτίνα στην οποία μπορεί να ταξιδέψει το φώς

```
PointLight {
        on TRUE
        intensity 1
        ambientIntensity 0
        color 1 1 1
        location 0 0 0
        attenuation 1 0 0
        radius 100
}
```

- Spot Light

Δημιουργεί έναν κώνο φωτός. Καθορίζει μια πηγή φωτός η οποία δείχνει προς μια συγκεκριμένη κατεύθυνση. Περιέχει τα ακόλουθα πεδία:

- on: Καθορίζει εάν το φώς είναι ενεργό
- intensity: Καθορίζει πόσο έντονο είναι το φώς και παίρνει τιμές οπό 0 έως 1
- ambientIntensity: Καθορίζει το ποσοστό συμμετοχής στο συνολικό φωτισμό και παίρνει τιμές από 0 έως 1
- color: είναι ένα RGB πεδίο που καθορίζει το χρώμα του φωτός
- location: Καθορίζει την θέση του φωτός με ένα 3D διάνυσμα
- direction: Καθορίζει την κατεύθυνση του φωτός με ένα 3D διάνυσμα
- attenuation: Είναι ένα 3D διάνυσμα το οποίο καθορίζει τον τρόπο με τον οποίο το φώς χάνει την ένταση του καθώς απομακρυνόμαστε από την πηγή
- radius: Καθορίζει την μέγιστη ακτίνα στην οποία μπορεί να ταξιδέψει το φώς
- cutOffAngle: Καθορίζει τον κώνο μέσα στον οποίο κατασκευάζονται οι ακτίνες φωτός. Θα πρέπει να είναι μεγαλύτερο ή ίσο του μηδενός και μικρότερο ή ίσο των $180^{o}$
- beamWidth: Καθορίζει έναν εσωτερικό κόμβο μέσα στον οποίο οι ακτίνες φωτός έχουν ίδια ένταση. Θα πρέπει να είναι μεγαλύτερο ή ίσο του μηδενός και μικρότερο ή ίσο των $180^{o}$

```
SpotLight {
        on TRUE
        intensity 0.3
        ambientIntensity 0
        color 0.89 0.15  0.21
        location 0 0 0
        direction 1 0 0
        attenuation 0 0 0
        radius 100
        cutOffAngle 0.78
        beamWidth 1.57
}
```

## 5.5 Ομαδοποίηση κόμβων

Ένα σύνολο κόμβων ορίζεται στην VRML ως group. Στην παρούσα εργασία έχουν χρησιμοποιηθεί οι ακόλουθοι κόμβοι ομαδοποίησης:

- Collision: Ορίζει ένα σύνολο κόμβων για τους οποίους ο browser ενημερώνεται όταν υπάρχει σύγκρουση

```
DEF table1 Transform{

        children[
                Collision{
                        children[

                                DEF t1 Transform{
                                ……..
                                }
                                DEF t2 Transform {
                                ……..
                                }
                        ] collide TRUE
                }
        ]
}
```

- Group: Ορίζει ένα καινούριο κόμβο ορισμένο από ένα σύνολο άλλων κόμβων έτσι ώστε να μπορεί να επαναχρησιμοποιηθεί αργότερα χωρίς να επαναληφθούν όλοι οι επιμέρους κόμβοι. Επίσης χρησιμοποιείται για εφαρμογή κάποιων γεγονότων (π.χ. animation, φωτισμός), σε συγκεκριμένα αντικείμενα τα οποία περιλαμβάνονται στην ομαδοποίηση

```
Group {
        children [
                DEF t1 Transform{
                …….
                }
                DEF t2 Transform {
                …….
                }
        ]
}
```

- Transform: Αλλάζει τις συντεταγμένες των αντικειμένων ώστε αυτά να τοποθετηθούν σε άλλο σημείο του κόσμου χωρίς αλλαγή των αρχικών τους συντεταγμένων. Μπορεί να χρησιμοποιηθεί για την εκτέλεση των παρακάτω μετασχηματισμών:
    - Scale: Αλλαγή κλίμακας
    - Rotation: Περιστροφή
    - Translation: Αλλαγή συντεταγμένων θέσης

```
DEF t1 Transform{
        scale 0.3 0.3 0.3
        rotation 1 0 0 1.6
```

```
translation  40 -8.6 -50
center -7 0 0
        children [
        ……..
        ]
}
```

**Routes**
Είναι ένας απλός τρόπος καθορισμού ενός μονοπατιού ανάμεσα σε ένα γεγονός που παράγεται από έναν κόμβο και ενός κόμβου που λαμβάνει την ενέργεια.

```
ROUTE Door1lock_sensor.touchTime TO door_lock.set_startTime
```

## 5.6 Υλοποίηση γεγονότων βασισμένων σε timers και σε ενέργειες του ήρωα

**Ο κόμβος TimerSensor**
Ο κόμβος TimerSensor είναι ένα ρολόι το οποίο παράγει γεγονότα καθώς περνάει ο χρόνος. Αυτά τα γεγονότα μπορεί παραδείγματος χάριν να είναι κάποια animations. Ο κόμβος περιλαμβάνει τα ακόλουθα πεδία:

- enabled: Καθορίζει την κατάσταση του sensor
- startTime: Καθορίζει το πότε αρχίζει να παράγει γεγονότα ο sensor
- stopTime: Καθορίζει τι πότε σταματάει να παράγει γεγονότα ο sensor
- cycleInterval:Καθορίζει τον αριθμό των δευτερολέπτων για τον οποίο ο sensor παράγει γεγονότα
- loop: Καθορίζει εάν τα γεγονότα τα οποία παράγει ο sensor είναι επαναλαμβανόμενα ή όχι

```
DEF timer_door1Open TimeSensor {
        cycleInterval 2
        loop FALSE
}
```

**Ο κόμβος TouchSensor**
Ο κόμβος TouchSensor είναι ένας τρόπος αλληλεπίδρασης με τον ήρωα. Καθορίζεται με τη βοήθεια του κόμβου group και επηρεάζει όλα τα αντικείμενα μέσα σε αυτό. Ενεργοποιείται όταν ο χρήστης τοποθετεί το ποντίκι πάνω στα αντικείμενα και όταν ο χρήστης «πατήσει» το αντικείμενο. Ο sensor περιέχει ένα μόνο πεδίο, ο οποίο δηλώνει εάν είναι ενεργοποιημένος ή όχι.

```
DEF Door1lock_sensor TouchSensor {}

ROUTE Door1lock_sensor.touchTime TO door_lock.set_startTime
```

**Dragging Sensors**
Οι Dragging Sensors αναγνωρίζουν την κίνηση του χρήστη, αλλά μπορούν επίσης να μετακινήσουν αντικείμενα τα οποία βρίσκονται στο ίδιο group. Για τους σκοπούς της παρούσας εργασίας, έχει χρησιμοποιηθεί μόνο ο CylinderSensor:

- CylinderSensor: Ακολουθεί την κίνηση ενός υποθετικού κυλίνδρου, αλλάζοντας τις συντεταγμένες των αντικειμένων ως προς τον y άξονα. Εκτελεί την περιστροφή βασιζόμενος σε δύο γωνίες. Περιέχει τα ακόλουθα πεδία:
    - enabled: Ορίζει εάν είναι ενεργός ή όχι
    - maxAngle: Καθορίζει τη μέγιστη περιστροφή

- minAngle: Καθορίζει την ελάχιστη περιστροφή

```
DEF cs CylinderSensor {
        enabled TRUE
       minAngle 0
       maxAngle 1.00
}

ROUTE cs.rotation_changed TO Door4_a.set_rotation
```

**Ο κόμβος ProximitySensor**

Είναι και αυτός ένας τρόπος αλληλεπίδρασης με τον ήρωα. Παράγει γεγονότα όταν ο ήρωας μπαίνει, βγαίνει ή κινείται σε μία συγκεκριμένη τετράγωνη περιοχή. Περιέχει τα ακόλουθα πεδία:

- enabled: Καθορίζει την κατάσταση του sensor
- center: Καθορίζει το κέντρο της τετράγωνης περιοχής
- size: Καθορίζει το μέγεθος της τετράγωνης περιοχής

```
DEF     proximity_table ProximitySensor {
         size 50 30 20
         center 0 0 0
          enabled TRUE
}

ROUTE proximity_table.enterTime TO proximity_timer.set_startTime
```

**Interpolators**

Οι Interpolators καθορίζουν ένα κλειδί (key) το οποίο κυμαίνεται από 0 έως 1 και μια τιμή κλειδιού (keyValue) οποία μπορεί να πάρει διάφορους τύπους τιμών, ανάλογα με τον Interpolator. Όλα τα πεδία και τα γεγονότα είναι ίδια για όλους τους Interpolators και μόνο οι τύποι των δεδομένων αλλάζουν. Στην παρούσα εργασία έχουμε χρησιμοποιήσει τους ακόλουθος Interpolators:

- ColorInterpolator: Λαμβάνει μια λίστα από RGB τιμές στο πεδίο KeyValue

```
DEF ci ColorInterpolator {
        key [ 0 0.3 0.6 1 ]
        keyValue [0.85 0.85 0.1, 0.251 0.8784 0.8157, 0.53 0.15 0.34, 0.03 0.18
        0.33]
        }

ROUTE color_sensor.fraction_changed TO ci.set_fraction
ROUTE ci.value_changed TO crossmat.set_diffuseColor
```

- OrientationInterpolator: Λαμβάνει μία λίστα από τιμές περιστροφής στο πεδίο KeyValue

```
DEF chess_oi OrientationInterpolator {
        key [ 0 1 ]
        keyValue [ 1 0 0 0, 1 0 0 -1.57]
}
ROUTE chess_oi.value_changed  TO top.set_rotation
```

- PositionInterpolator: Λαμβάνει μία λίστα από 3D συντεταγμένες στο πεδίο KeyValue

```
DEF pi_Hidden_map PositionInterpolator {
        key [ 0 1 ]
        keyValue [0 -3 0, 0 -2 0]
}

ROUTE pi_Hidden_map.value_changed TO Hidden_map.set_translation
```

## 5.7 Ήχοι

Με τη χρήση του Sound node καθορίζουμε την θέση του ήχου καθώς και τις γεωγραφικές ιδιότητες για την διάδοση του. Περιλαμβάνει τα ακόλουθα πεδία:

- location: Καθορίζει την θέση της πηγής του ήχου στο σύστημα συντεταγμένων
- intensity: Καθορίζει την ένταση του ήχου
- direction: Καθορίζει την κατεύθυνση προς την οποία κατευθύνεται ο ήχος
- priority: Καθορίζει την προτεραιότητα του ήχου. Υψηλότερες τιμές καθορίζουν μεγαλύτερη προτεραιότητα
- spatialize: Καθορίζει εάν ο ήχος θα αναγνωριστεί ως 3D ήχος ή σαν περιβαλλοντικός ήχος. Εάν είναι αληθής τότε ο ήχος θεωρείται 3D, διαφορετικά ο ήχος ακούγεται τόσο στο αριστερό όσο και στο δεξί κανάλι ανεξάρτητα από τη θέση του ήρωα
- minBack, minFront: Καθορίζει το εσωτερικό ελλειψοειδές μέσα στο οποίο ακούγεται ο ήχος στην ένταση που έχει καθοριστεί
- maxBack, maxFront: Καθορίζει το εξωτερικό. Εάν ο ήρωας βρίσκεται έξω από αυτό, ο ήχος δεν ακούγεται

```
Sound {
        intensity 100
        priority 0
        spatialize TRUE
        minBack 1000
        minFront 1000
        maxBack 1000
        maxFront 100
        source
                DEF door_lock AudioClip{
                        loop FALSE
                        url "door_lock_1.wav"
                }
        location 0 0 0 }

ROUTE Door1Open_sensor.touchTime TO door_lock.set_startTime
```

## 5.8 NavigationInfo

Ο κόμβος NavigationInfo περιγράφει τον ήρωα και καθορίζει τις ιδιότητες πλοήγησης. Περιέχει τα ακόλουθα πεδία:

- avatarSize: Καθορίζει τις φυσικές διαστάσεις του ήρωα. Είναι χρήσιμος για collision detaction
- headlight: Είναι μία Boolean μεταβλητή, η οποία καθορίζει εάν το headlight είναι ενεργό ή όχι
- visibilityLimit: Καθορίζει τη μέγιστη απόσταση στην οποία μπορεί να δει ο ήρωας
- speed: Καθορίζει την ταχύτητα του χρήστη, σε μέτρα το δευτερόλεπτο

- type: Καθορίζει τον τρόπο πλοήγησης του ήρωα

```
NavigationInfo {
        avatarSize [2,8, 2]
        headlight TRUE
        speed 1.5
        type "ANY"
        visibilityLimit 400
}
```

## 5.9 Ο κόμβος ViewPoint

Καθορίζει την θέση την θέση του ήρωα καθώς και την ορατότητα του. Αποτελείται από τα ακόλουθα πεδία:

- fieldOfView: Καθορίζει μια γωνία σε ακτίνια.
- position: Καθορίζει την θέση του ήρωα στο κόσμο
- orientation: Καθορίζει την κατεύθυνση προς την οποία κοιτάζει ο ήρωας
- description: Αποτελεί μια περιγραφή του viewpoint
- jump: Ορίζει εάν μπορεί να υπάρξει μετάβαση σε αυτό το Viewpoint, κατά τη μετάβαση από ένα Viewpoint σε άλλο

```
DEF CAMERA1 Viewpoint {
        fieldOfView 2
        position  12  -4 -20
        orientation          0 1 0 -0.4
        description "room1-start"
        jump FALSE
}
```

## 5.10 Ο κόμβος Background

Παρέχει έναν τρόπο ορισμού του ορίζοντα του κόσμου. Μας επιτρέπει να ορίσουμε τον ουρανό, και το έδαφος.

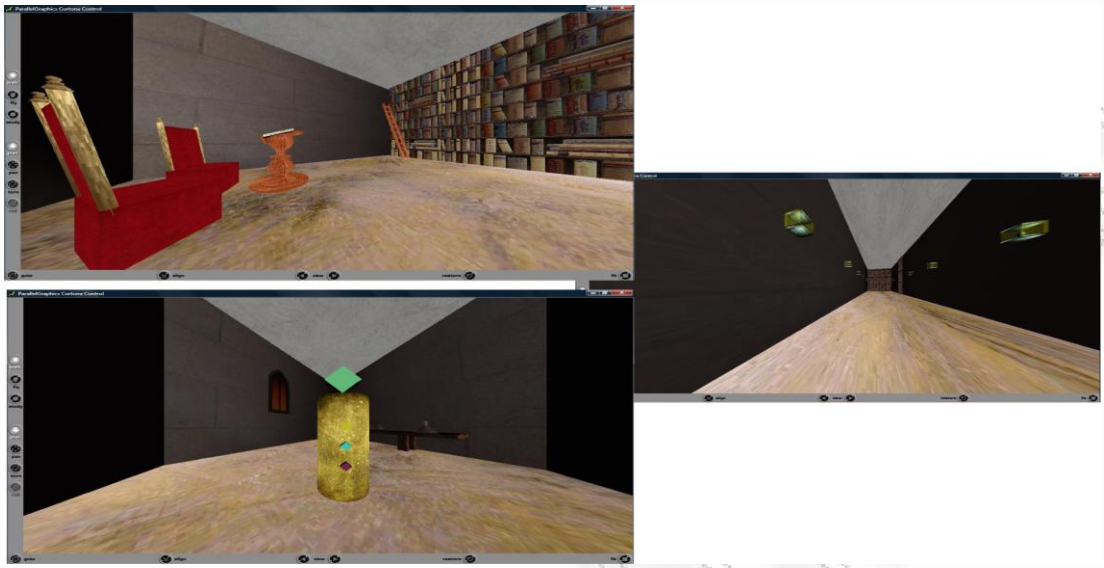
```
Background {
        groundColor        [0.5 0.5 0.5, 0 0 0]
        groundAngle  [ 1.57]
        skyColor [0 1 1, 1 0 1, 1 0 0]
        skyAngle [0.78, 1.57]
}
```

## 5.11 Ο κόμβος Fog

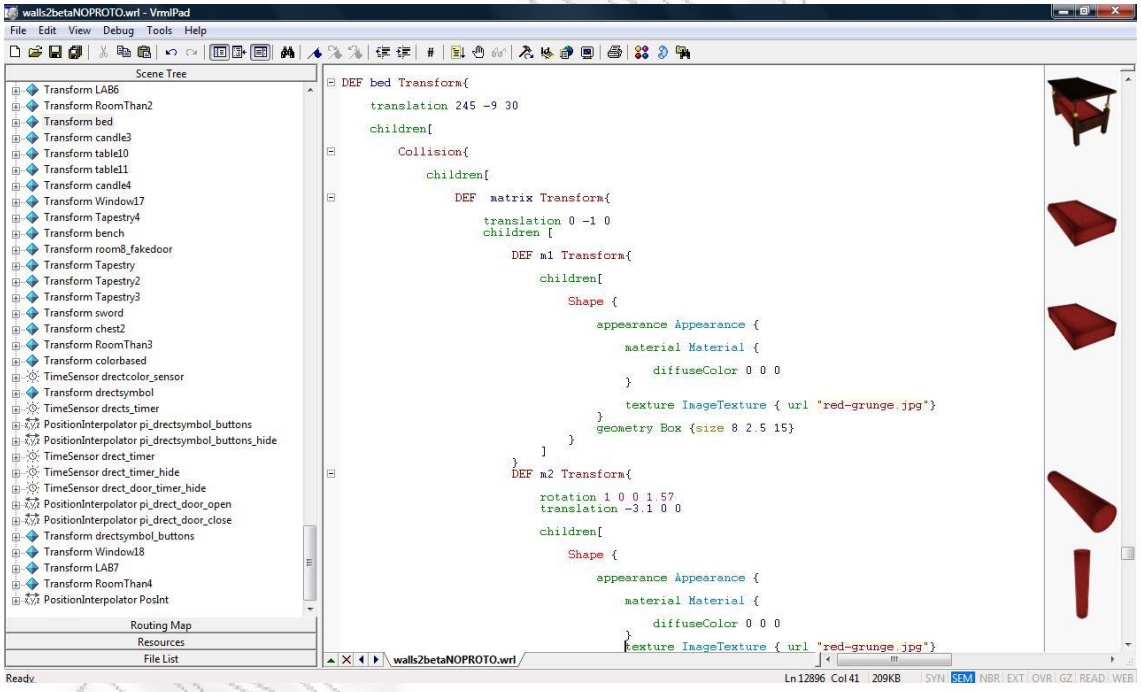
Χρησιμοποιείται για να προσθέσει ρεαλισμό στον κόσμο μας. Παρέχει ατμόσφαιρα, δημιουργώντας ομίχλη. Περιέχει τα ακόλουθα πεδία:

- color: Καθορίζει το χρώμα της ομίχλης
- visibilityRange: Καθορίζει τη μέγιστη απόσταση στην οποία μπορεί να δει ο ήρωας

```
Fog {
        color 0.05 0.03 0.03
        visibilityRange 60
}
```



**Δείγματα του εικονικού κόσμου που δημιουργήθηκε με χρήση της VRML**




**VrmlPad**

# 6. Αναπαράσταση του κόσμου σε VERL

Αφού έχουμε υλοποιήσει τον αρχικό μας κόσμο σε VRML επόμενο βήμα είναι η αναπαράσταση του στη γλώσσα VERL. Σε αυτό το κεφάλαιο θα περιγραφούν οι τεχνικές και οι δυνατότητες οι οποίες χρησιμοποιήθηκαν για τις ανάγκες της παρούσας εργασίας από την γλώσσα VERL καθώς και οι αλλαγές οι οποίες χρειάστηκε να πραγματοποιηθούν στον αρχικό κόσμο υλοποιημένο σε VRML προκειμένου να είναι λειτουργικός ο κόσμος και να υποστηρίζονται οι δυνατότητες της αναπαράστασης REVE.

## 6.1 Virtual Model Definition

### 6.1.1 Ορισμός

Αρχικά ορίζουμε το path για το DTD αρχείο στο οποίο ορίζεται η γλώσσα VERL.

```
<!-- this file requires the DTD to be at " C:/Program Files/REVEWorlds-1.43.5-informatics_unipi/verl-1.0.dtd", for other locations and platforms, adjust the following path accordingly -->
<!DOCTYPE abr SYSTEM "file:///C:/Program Files/REVEWorlds-1.43.5-informatics_unipi/verl-1.0.dtd">
```

Στη συνέχεια ανοίγουμε το tag “world” μέσα στο οποίο θα ορίσουμε όλα τα items με τις ιδιότητες. Στο tag “world” μπορούμε να ορίσουμε χαρακτηριστικά όπως η version του κόσμου, το όνομα του και οι διαστάσεις του. Επίσης σε αυτό το σημείο μπορούμε να ορίσουμε και το tag “info” στο οποίο μπορούμε εφόσον θέλουμε να καταχωρίσουμε κάποια στοιχεία για το πρόγραμμα μας όπως είναι ο συντάκτης, τα στοιχεία του η ημερομηνία και η έκδοση.

```
<abr version="1.0">

 <world version="5.0" name="castle" dimensions="">

  <info
   author="Artemis Loukritia Apostolou"
   date="11/01/2011"
   revision="1"
   email="artemis_loukritia@yahoo.gr"
   organization="University of Piraeus"
   description="Castle"
  />
……..
……..
……..

 </world>

</abr>
```

Επόμενο βήμα είναι ο ορισμός ενός itemgroup μέσα στο οποίο θα συμπεριληφθούν τα διάφορα items του κόσμου. Για κάθε itemgroup είναι υποχρεωτικό να ορίσουμε ένα όνομα, καθώς και το path για το vrml αρχείο (.wrl) στο οποίο ορίζονται τα items τα οποία θα συμπεριληφθούν στο itemgroup. Επίσης ορίζουμε εάν θα κληρονομηθεί στο REVE Worlds η συμπεριφορά των αντικειμένων όπως μπορεί να ορίζεται στην VRML (μέσω π.χ. από κάποιους interpolators) ορίζοντας ανάλογα με το επιθυμητό αποτέλεσμα το tag “behaviours” με τιμή true ή false αντίστοιχα.

```
<itemGroup name="walls" location="models/castle/walls/walls.wrl" behaviours="false">
    ……
    ……
    ……
</itemGroup>
```

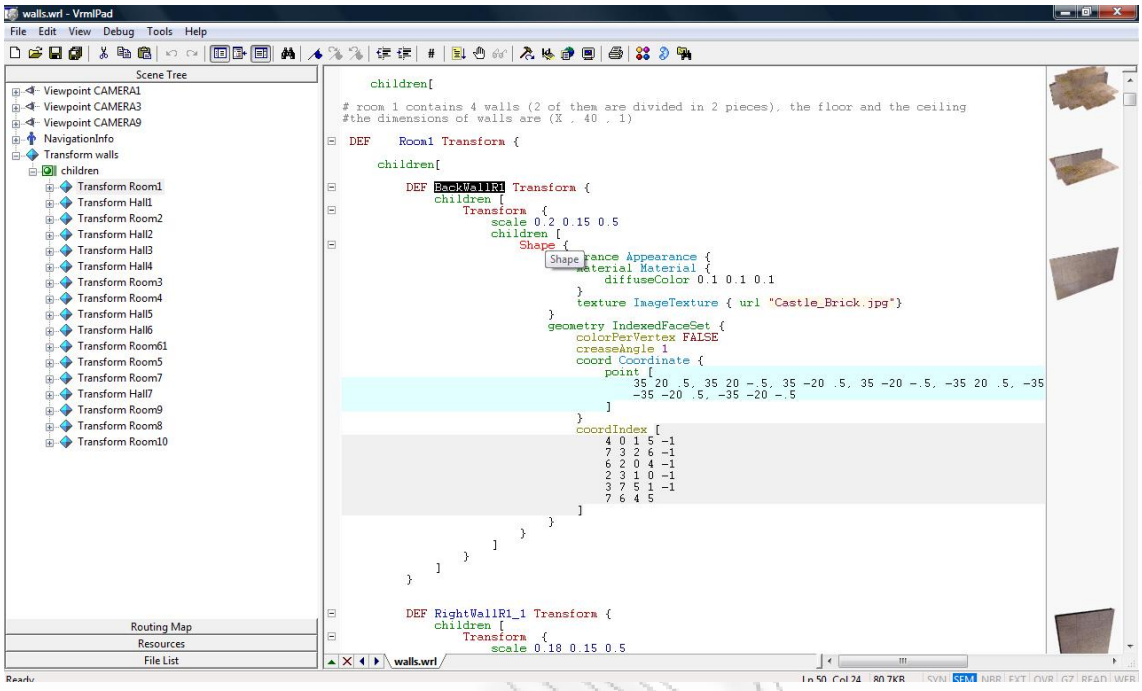
Στη συνέχεια ορίζουμε τα αντικείμενα μας όπου αρχικά δίνουμε ένα όνομα σε αυτά. Σημαντικό βήμα σε αυτό το σημείο είναι ο ορισμός της κλάσης του αντικειμένου όπου μπορεί να είναι real ή unreal. Όταν ένα αντικείμενο οριστεί ως real τότε το σύστημα υπολογίζει το bounding box του και έτσι ένας πράκτορας μπορεί να το αντιληφθεί. Ουσιαστικά ένας πράκτορας συγκεντρώνει πληροφορίες για τον κόσμο και από αυτές τις πληροφορίες εξάγει συμπεράσματα. Με αυτόν τον τρόπο μπορεί και να κινηθεί ή να "δει" αντικείμενα στον κόσμο. Για κάθε αντικείμενο αυτό που μπορεί να καταλάβει είναι το bounding box του και έτσι υπολογίζει τον κενό χώρο ο οποίο υπάρχει στον κόσμο και μέσα στον οποίο μπορεί να κινηθεί. Έτσι παραδείγματος χάρη εάν έχουμε ένα δωμάτιο σε vrml κατά την απεικόνιση του για το REVE Worlds θα πρέπει να ορίσουμε κάθε τοίχο του ξεχωριστά έτσι ώστε ο πράκτορας να δει ξεχωριστά bounding boxes για κάθε τοίχο και να θεωρήσει τον χώρο ανάμεσα τους ως κενό. Αν ορίζαμε όλο το δωμάτιο ως ένα item τότε αυτό θα περικλειόταν από ένα και μοναδικό bounding box και ο πράκτορας θα το εκλάμβανε ως ένα συμπαγές αντικείμενο. Ορίζοντας ένα αντικείμενο ως unreal το σύστημα δεν υπολογίζει καθόλου bounding boxes και ο πράκτορας ουσιαστικά δεν βλέπει τα αντικείμενα αυτά. Θα πρέπει αν τονιστεί σε αυτό το σημείο ότι οι επιφάνειες πάνω στις οποίες κινείτε ο πράκτορας θα πρέπει να ορίζονται ως unreal γιατί διαφορετικά εγκλωβίζεται μέσα σε αυτό το bounding box με αποτέλεσμα να μην μπορεί να βγει.

```
<item name="BackWallR1">

    <virtualModel source="BackWallR1">
        <transform translation="0, 0, 13.5" />
    </virtualModel>

</item>
```

```
<item name="FloorR1" class="unreal">

    <virtualModel source="FloorR1">
        <transform translation="0, -0.5 0"/>
    </virtualModel>

</item>
```

Άλλα χαρακτηριστικά τα οποία ορίζονται σε αυτό το σημείο είναι αν το item θα είναι fit ή fitcenter. Εάν ένα item είναι fit τότε το τοπικό σύστημα συντεταγμένων του ανατίθεται στο σύστημα συντεταγμένων του κόσμου, ενώ αν είναι και fitcenter τότε και το σύστημα συντεταγμένων του bounding box του θα εναρμονιστεί με το σύστημα συντεταγμένων του κόσμου.

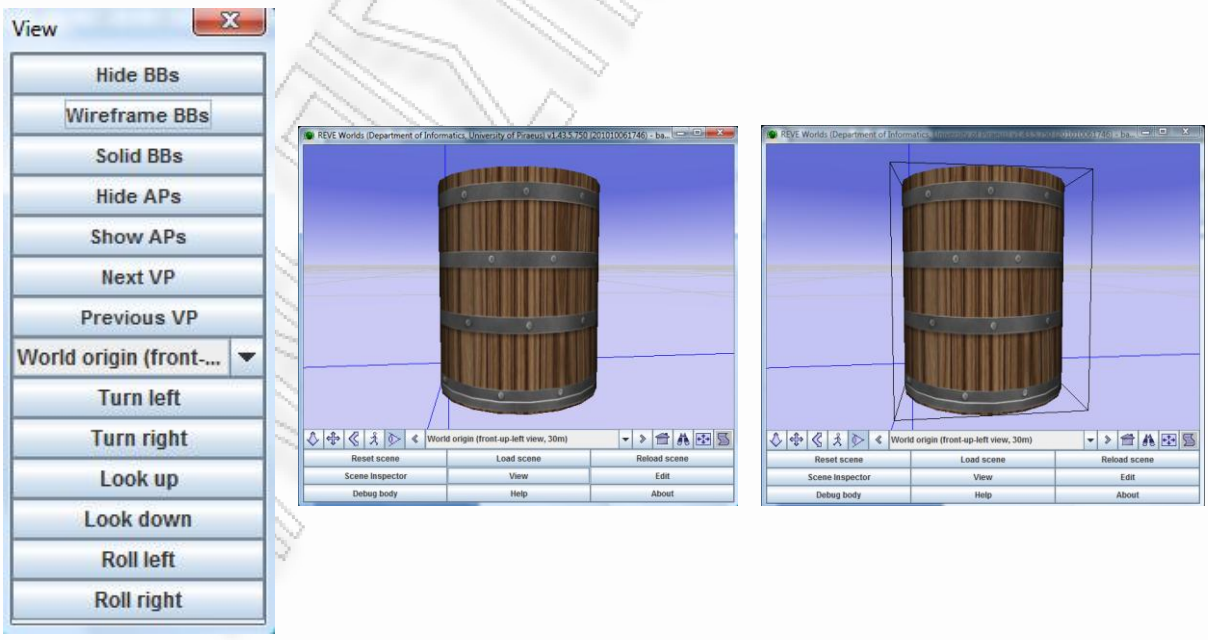
Μέσα στο tag "item" θα ορίσουμε και το virtual model θα αποδώσουμε δηλαδή ουσιαστικά τα χαρακτηριστικά του physical aspect. Ορίζουμε το tag "virtualModel" όπου αρχικά πρέπει να υποδείξουμε με στο source τον αντίστοιχο κόμβο του item που θέλουμε στο vrml αρχείο. Τέλος στο tag "virtualModel" ορίζουμε ένα tag "transform" στο οποίο ορίζουμε το translation, το rotation και το scale του κάθε item.




**VRML source node**

### 6.1.2 Προβλήματα Υπολογισμού Bounding Boxes

Το REVE Worlds δίνει στον χρήστη την επιλογή της οπτικής απεικόνισης των υπολογιζόμενων bounding boxes αλλά και επιπρόσθετα στοιχεία για αυτά όπως το μέγεθος του. Έπειτα όμως από τον αρχικό ορισμό σε VERL των items σύμφωνα με τα παραπάνω παρατηρήθηκε ότι δεν ήταν δυνατός ο υπολογισμός των bounding boxes όπως και θα έπρεπε.




**REVE Worlds bounding boxes**

Πρώτο μας βήμα είναι η αφαίρεση των κόμβων Collision οι οποίοι κάποιες φορές δημιουργούν πρόβλημα στον υπολογισμό των bounding boxes. Η αφαίρεση αυτών των κόμβων δεν μας επηρεάζει

καθόλου στην παρούσα εφαρμογή καθώς αφορούν αλληλεπίδραση του χρήστη με τον κόσμο (navigation χρηστών) και ουσιαστικά δεν προσθέτουν κάποια επιπλέον πληροφορία. Έτσι όλα τα αντικείμενα του κόσμου θα πρέπει να τροποποιηθούν σύμφωνα με το ακόλουθο παράδειγμα.

```
DEF Barrels Transform{
        children[
                Collision{
                        children[
                                DEF barrel1 Transform {
                                        translation 0 0 0
                                        children [
                                                Shape {
                                                        appearance Appearance {
                                                                material Material {
                                                                        diffuseColor 1 1 1
                                                                }
                                                                texture ImageTexture {
                                                                        url "barrell.jpg"
                                                                }
                                                        }
                                                        geometry Box {size 1 1 1}
                                                }
                                        ]
                                }
                        ]
                }
        ]
}
```

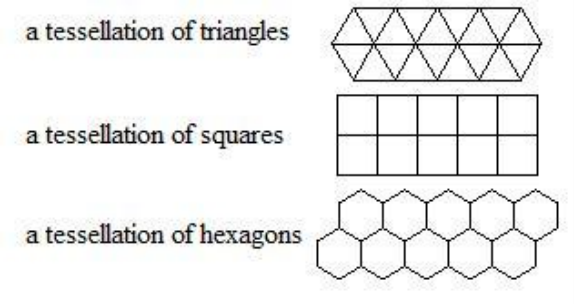
**Αρχικός κόμβος με Collision**

```
DEF Barrels Transform{
    children[
        DEF barrel1 Transform {
            translation 0 0 0
            children [
                Shape {
                    appearance Appearance {
                        material Material {
                            diffuseColor 1 1 1
                        }
                        texture ImageTexture { url "barrell.jpg"}
                    }
                    geometry Box {size 1 1 1}
                }
            ]
        }

    ]
}
```

**Τροποποιημένος κόμβος χωρίς Collision**

Έπειτα από την αφαίρεση των κόμβων Collision το πρόβλημα του υπολογισμού των bounding boxes παραμένει. Ο λόγος για τον οποίο αυτό συμβαίνει είναι γιατί βάσει X3D/VRML specification δεν είναι δυνατή η εξαγωγή πληροφορίας για γεωμετρικά αντικείμενα τύπου sphere, cylinder, box και cone από την στιγμή όπου έχει ξεκινήσει το rendering της σκηνής οπότε είναι αδύνατο να υπολογιστεί το bounding box. Έτσι από τη στιγμή που εισαχθεί στη σκηνή ένα αντικείμενο τύπου π.χ. box, είναι αδύνατο να διαβαστεί η τιμή του πεδίου size.

Για να επιλυθεί αυτό το πρόβλημα θα πρέπει να μετατραπούν όλα τα παραπάνω geometry primitives σε IndexFacedSets. Ένα IndexedFacedSet ουσιαστικά επιτρέπει τον ορισμό των γεωμετρικών ιδιοτήτων των αντικειμένων μέσω ενός συνόλου πολυγώνων τα οποία σχηματίζουν το αρχικό γεωμετρικό αντικείμενο. Η τεχνική αυτή είναι γνωστή ως tessellation, η δημιουργία δηλαδή ενός σχήματος από ένα σύνολο ίδιων μεταξύ τους πολυγώνων τα οποία επαναλαμβάνονται συνεχώς χωρίς να δημιουργείτε κενός χώρος και χωρίς αυτά να επικαλύπτονται.




**Tesselations(πηγή The Math Forum)**

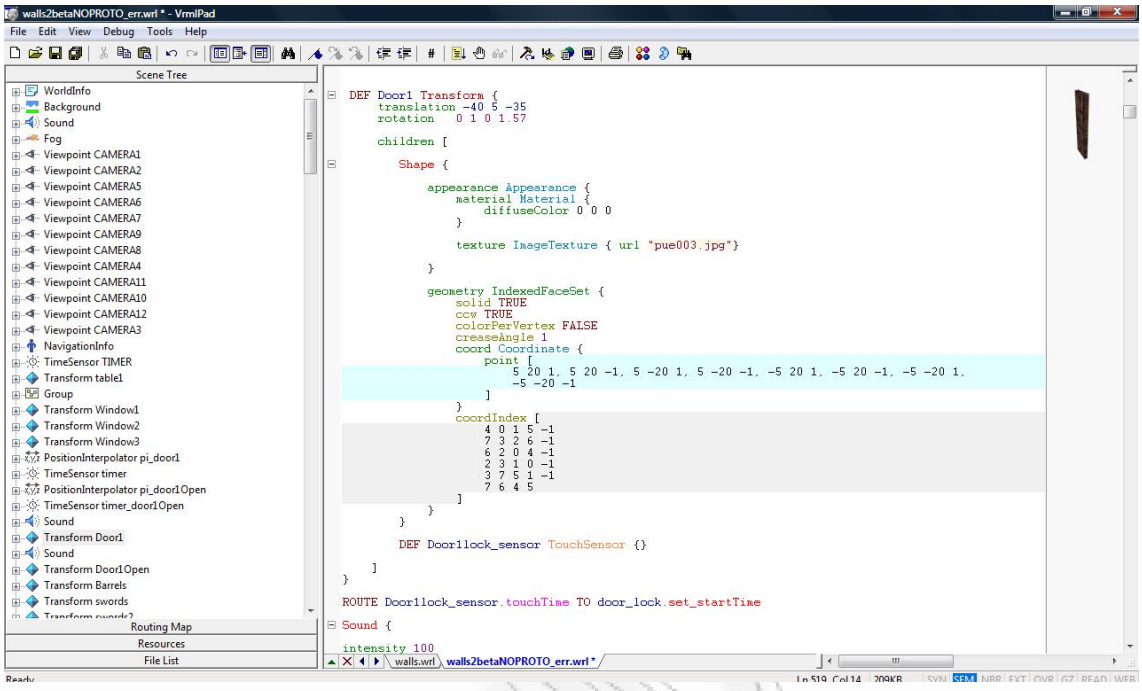
Σε ένα IndexedFacedSet το καθένα από τα επιμέρους πολύγωνα καθορίζεται από μια λίστα στην οποία περιέχονται οι συντεταγμένες στον τρισδιάστατο χώρο των κορυφών του, και η οποία αποθηκεύεται στον κόμβο Coordinate. Ο κόμβος αυτός περνάει ως όρισμα στο πεδίο coord. Επιπλέον σε μια άλλη λίστα

αποθηκεύονται οι κορυφές του κάθε πολυγώνου (πεδίο coordIndex). Επιπλέον ορίσματα είναι το solid με τιμές true ή false που καθορίζει το εάν ένας browser μπορεί να σχεδιάσει και τις δύο όψεις ενός πολυγώνου, το ccw με τιμές true ή false που καθορίζει το εάν τα σημεία που καθορίζουν ένα πολύγωνο θα εφαρμοστούν αριστερόστροφα ή δεξιόστροφα αντίστοιχα, το colorPerVertex με τιμές true ή false που καθορίζει πως θα εφαρμοστούν τα χρώματα και το creaseAngle στο οποίο ορίζεται ένα κατώφλι για τις γωνίες που δημιουργούνται από την ένωση δυο γειτονικών πολυγώνων. Αν η γωνία αυτή είναι μεγαλύτερη από το κατώφλι τότε θα φαίνεται καθαρά το σημείο ένωσης των δύο πολυγώνων ενώ σε διαφορετική περίπτωση η τομή αυτή θα είναι εξομαλυσμένη.

```
geometry IndexedFaceSet {
        solid TRUE
        ccw TRUE
        colorPerVertex FALSE
        creaseAngle 1
        coord Coordinate {
                point [5 20 1, 5 20 -1, 5 -20 1, 5 -20 -1, -5 20 1, -5 20 -1, -5 -20 1,-5 -20 -1]
        }
        coordIndex [
                4 0 1 5 -1
                7 3 2 6 -1
                6 2 0 4 -1
                2 3 1 0 -1
                3 7 5 1 -1
                7 6 4 5
        ]
}
```

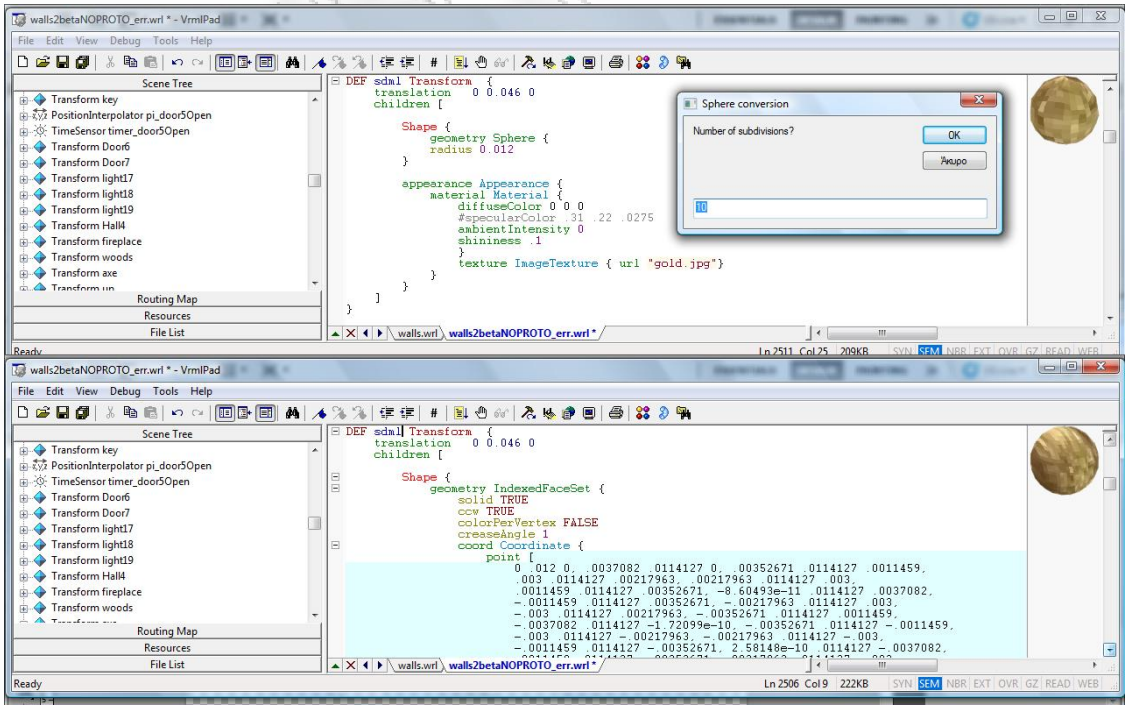
Η μετατροπή αυτή γίνεται μέσω του VRMLPad editor ο οποίος δίνει την επιλογή για γεωμετρίες τύπου box, cylinder και cone της μετατροπής αυτής.


**Μετατροπή σε IndexedFaceSet μέσω VRML Pad**




**Box to IndexedFaceSet**

Για τα γεωμετρικά αντικείμενα τύπου sphere χρησιμοποιήθηκε ένα plugin συντάκτης του οποίου είναι ο κ. Γιώργος Αναστασσάκης, το MadVrmlPad.bas το οποίο ενσωματώνει τις δυνατότητες του SAMPLE.bas του directory "AddIns" του VRMLPad. Για την χρήση αυτού του extra plugin αφαιρέθηκε το απλό SAMPLE.bas και αντικαταστάθηκε από το καινούριο plugin. Το MadVrmlPad.bas επιπλέον επιτρέπει το tessellation των σφαιρών και δίνει στον χρήστη την επιλογή να ορίσει τα subdivisions, δηλαδή τις κάθετες και οριζόντιες διαιρέσεις που θα υποστεί η σφαίρα. Στην περίπτωση μας χρησιμοποιήθηκαν subdivisions της τάξης του 10.



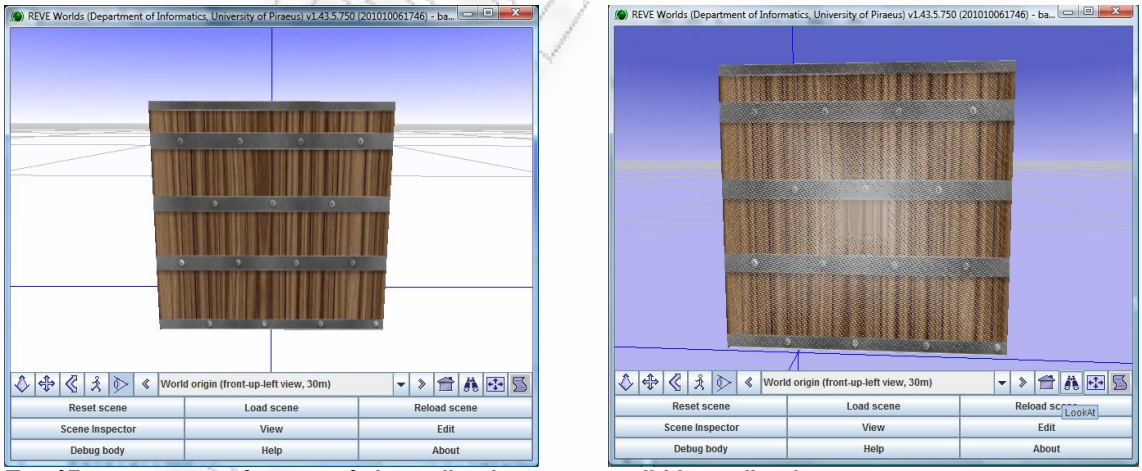

**Sphere to IndexedFaceSet**

Έτσι ένα αντικείμενο τύπου geometry box θα πρέπει να μετατραπεί σύμφωνα με το παρακάτω παράδειγμα:

| | |
|---|---|
| geometry Box {size 1 1 1} | geometry IndexedFaceSet {<br>solid TRUE<br>ccw TRUE<br>colorPerVertex FALSE<br>creaseAngle 1<br>coord Coordinate {<br>point [<br>.5 .5 .5, .5 .5 -.5, .5 -.5 .5, .5 -.5 -.5, -.5 .5 .5, -.5 .5 -.5,.5 -.5 .5, -.5 -.5 -.5<br>]<br>}<br>coordIndex [<br>4 0 1 5 -1<br>7 3 2 6 -1<br>6 2 0 4 -1<br>2 3 1 0 -1<br>3 7 5 1 -1<br>7 6 4 5<br>]<br>} |

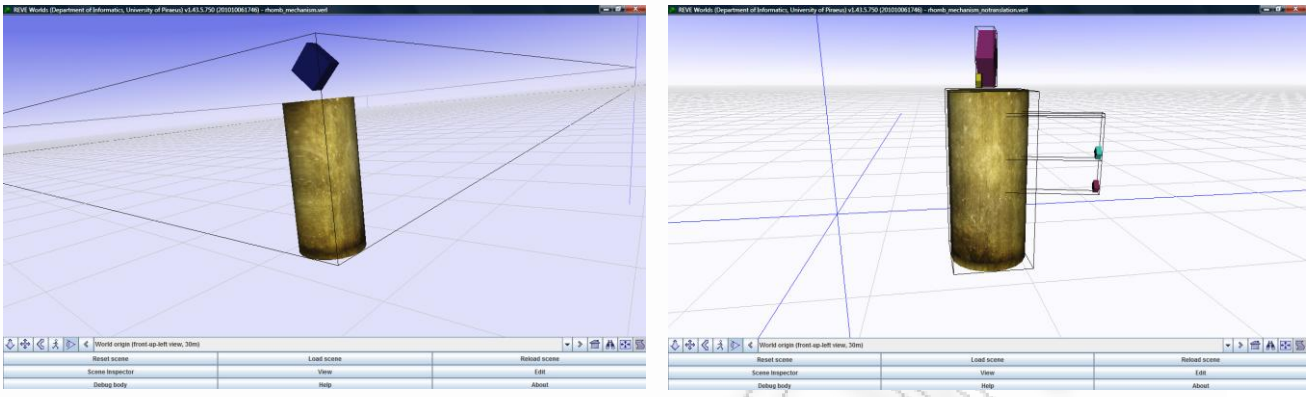
**Πηγαίος κώδικας μετατροπής geometry box σε IndexedFaceSet**

Με την παραπάνω αλλαγή είναι πλέον δυνατός ο υπολογισμός των bounding boxes των αντικειμένων. Ένα παράδειγμα φαίνεται στην παρακάτω εικόνα.




**Παράδειγμα αντικειμένου χωρίς bounding box και με solid bounding box**

Όμως παρατηρούμε ότι ενώ πλέον υπολογίζονται τα bounding boxes των items, ο υπολογισμός αυτός δεν είναι πάντα σωστός. Ένα παράδειγμα αποτελεί το παρακάτω αντικείμενο:



**Παραδείγματα λάθος υπολογισμού bounding box**

Ο λάθος αυτός υπολογισμός δεν οφείλεται σε κάποιο bug της πλατφόρμας η οποία δεν επιτρέπει τον σωστό υπολογισμό των bounding boxes αλλά σε κάτι το οποίο επηρεάζει το bounding box. Αυτός ο παράγοντας μπορεί πολλές φορές να είναι και αόρατος (τις περισσότερες φορές), και επηρεάζει την κατανομή της γεωμετρίας του bounding box. Για την εξακρίβωση της αιτίας κάθε φορά θα πρέπει να παρατηρήσουμε την υλοποίηση του κάθε item στη Vrml.

Για την πρώτη περίπτωση, παρατηρούμε ότι το item είναι δομημένο ως εξής:

```
DEF rhomb_mechanism Transform {
        children [
                DEF colorbaseb Transform{
                        Translation 80 0 -105
                                Children [
                                        DEF baseb Transform {
                                                translation -0 -0.5 7
                                                        children[
                                                                DEF colorbase Transform{
                                                                        Translation 310 -8 -167
                                                                                children[
……..
……..
                DEF     romssymbol_buttons Transform{
                        translation 383 -6 -258
                        rotation 0 1 0 1.57
                                children[
                                        DEF yellow_rom Transform{
                                                children[
                                                        DEF y_romsymba_but      Transform {
                                                                rotation 0 0 1 0.8
                                                                translation -0 -0.5 0
……..
……..
```

**Αρχικός πηγαίος κώδικας item με λάθος υπολογιζόμενο bounding box**

Παρατηρούμε ότι υπάρχουν πάρα πολλά translations (ίσως περισσότερα από ότι θα χρειάζονταν) τα οποία χρησιμοποιήθηκαν για την αρχική υλοποίηση σε VRML ώστε τα αντικείμενα να μετακινηθούν στην σωστή τους θέση. Αυτά τα translations τα οποία έχουν χρησιμοποιηθεί για αυτόν το σκοπό (και όχι αυτά που έχουν χρησιμοποιηθεί για να τοποθετηθούν οι κόμβοι του item στο σωστό σημείο) είναι και τελείως άχρηστοι κατά την μεταφορά του κόσμου στο REVE Worlds καθώς στον ορισμό του κόσμου σε Verl ορίζουμε εκ νέου το translation ενός item στο virtual model και δεν παίρνει πληροφορία για το translation από το .wrl αρχείο. Αντίθετα αυτή η πληροφορία από το .wrl αρχείο σχετικά με το translation εφόσον υπάρχει αναγκάζει το σύστημα να υπολογίσει το bounding box (σωστά από πλευράς του) με βάση το shape του item καθώς και το ορισμένο translation. Αφαιρώντας λοιπόν αυτή την πρόσθετη πληροφορία το bounding box υπολογίζεται σωστά.

```
DEF rhomb_mechanism Transform {
        children [
                Transform {
                        scale 0.2 0.2 0.2
                                children [
                                        DEF colorbaseb Transform{
                                                children[
                                                        DEF basebrhomb Transform {
                                                                children[
                                                                        DEF colorbaserhomb
                                                                        Transform{
                                                                                children[
……..
……..
                        DEF romssymbol_buttons Transform{
                                translation 0 5 0
                                rotation 0 1 0 1.57
                                children[
……..
……..
```

**Τελικός πηγαίος κώδικας item με λάθος υπολογιζόμενο bounding box**

Πάμε τώρα να εξετάσουμε την δεύτερη περίπτωση λανθασμένου υπολογισμού των bounding boxes. Και σε αυτή την περίπτωση συμβαίνει κάτι αντίστοιχο με την προηγούμενη περίπτωση, υπάρχει δηλαδή κάτι ουσιαστικά αόρατο, το οποίο επηρεάζει το bounding box. Αυτή την φορά δεν επηρεάζεται το bounding box ολόκληρου του αντικειμένου αλλά μόνο κάποιων κόμβων του.

```
DEF blue_rom Transform{
        children[
                DEF b_romsymba_but Transform {
                        rotation 0 0 1 0.8
                        translation 0 -2.5 7
                        children[
                                Shape {
                                        appearance Appearance{
                                                material Material {
```

```
                                        diffuseColor 0.251 0.8784 0.8157
                                }
                        }
                        geometry IndexedFaceSet {
                                solid TRUE
                                ccw TRUE
                                colorPerVertex FALSE
                                creaseAngle 1
                                coord Coordinate {
                                        point [
                                                .3 .3 .15, .3 .3 -.15, .3 -.3
                                                .15, .3 -.3 -.15, -.3 .3 .15,
                                                -.3 .3 -.15, -.3 -.3 .15, -.3
                                                -.3 -.15
                                        ]
                                }
                                coordIndex [
                                        4 0 1 5 -1
                                        7 3 2 6 -1
                                        6 2 0 4 -1
                                        2 3 1 0 -1
                                        3 7 5 1 -1
                                        7 6 4 5
                                ]
                        }
                }
        ]
    }
    DEF blue_rom_sensor TouchSensor {}
  ]
}
```

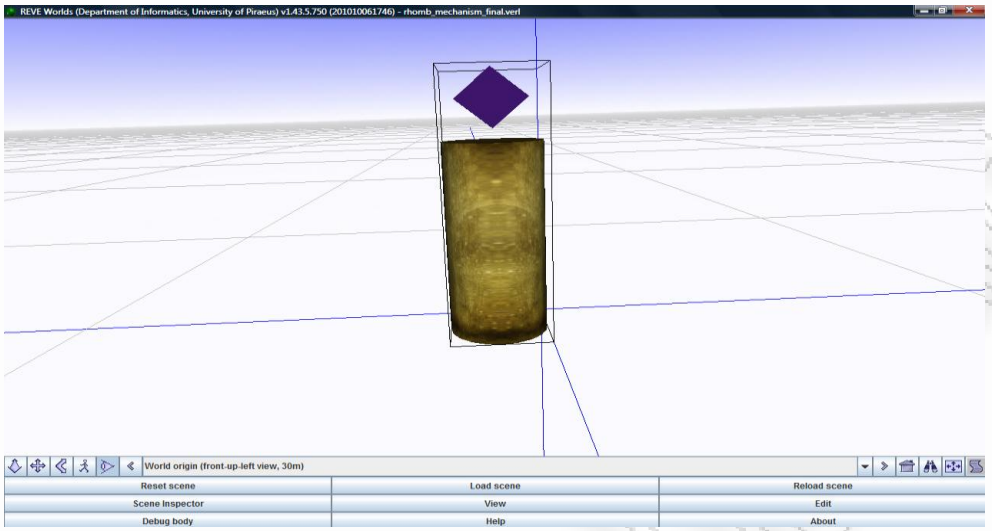
**Αρχικός πηγαίος κώδικας item με λάθος υπολογιζόμενο bounding box**

Παρατηρούμε ότι ο κόμβος blue_rom έχει δύο παιδιά, τον κόμβο b_romsymba_but και έναν TouchSensor. Καθώς ο b_romsymba_but ορίζει κάποιο translation, το αποτέλεσμα είναι να απομακρύνονται τα παιδιά του (συγκεκριμένα, το Shape) από τον κόμβο TouchSensor. Με άλλα λόγια, αντί να είναι και το TouchSensor και το Shape στην ίδια θέση, ανεξάρτητα με το ποια είναι αυτή, έχουν μεταξύ τους μια απόσταση όση ορίζει το translation του b_romsymba_but. Και αυτή τη φορά το σύστημα υπολογίζει από πλευράς του σωστά το bounding box του item με βάση τα στοιχεία που λαμβάνει.

Το πρόβλημα λύνεται εάν ορισθεί ορθότερα το item στη VRML έτσι ώστε ο κόμβος blue _rom να έχει ένα παιδί, τον κόμβο b_romsymba_but, ο οποίος με τη σειρά του θα έχει δύο παιδιά, τους κόμβους Shape και TouchSensor. Καθώς οι δύο τελευταίοι είναι παιδιά του ίδιου κόμβου Transform (του b_romsymba_but), η σχετική θέση τους δεν επηρεάζεται από το translation του. Έτσι, το σύστημα πολύ σωστά υπολογίζει bb με βάση τη γεωμετρία του Shape και μόνο, ανεξάρτητα από την τιμή translation του κόμβου b_romsymba_but.

```
DEF blue_rom Transform{
        children[
                DEF b_romsymba_but Transform {
                        rotation 0 0 1 0.8
                        translation 0 -2.5 7
                        children[
                                Shape {
                                        appearance Appearance{
                                                material Material {
                                                        diffuseColor 0.251 0.8784 0.8157
                                                }
                                        }
                                        geometry IndexedFaceSet {
                                                solid TRUE
                                                ccw TRUE
                                                colorPerVertex FALSE
                                                creaseAngle 1
                                                coord Coordinate {
                                                        point [
                                                                .3 .3 .15, .3 .3 -.15, .3 -.3
                                                                .15, .3 -.3 -.15, -.3 .3 .15,
                                                                -.3 .3 -.15, -.3 -.3 .15, -.3
                                                                -.3 -.15
                                                        ]
                                                }
                                                coordIndex [
                                                        4 0 1 5 -1
                                                        7 3 2 6 -1
                                                        6 2 0 4 -1
                                                        2 3 1 0 -1
                                                        3 7 5 1 -1
                                                        7 6 4 5
                                                ]
                                        }
                                }
                        DEF blue_rom_sensor TouchSensor {}
                        ]
                }
        ]
}
```

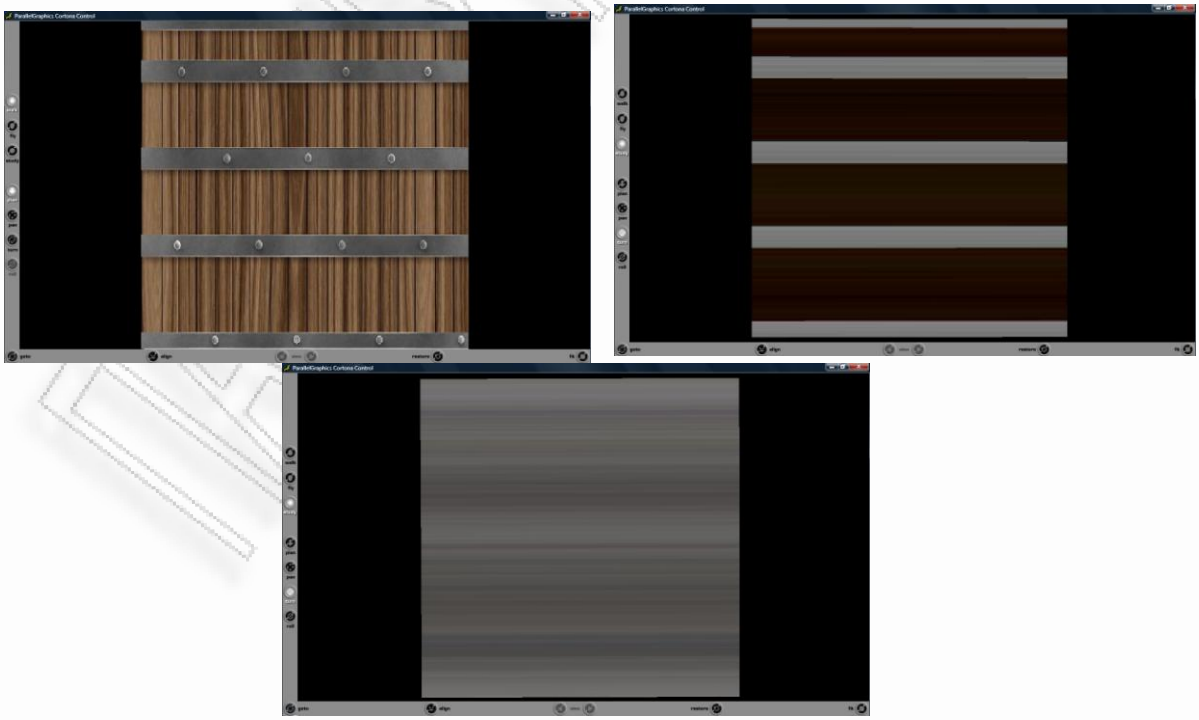
**Τελικός πηγαίος κώδικας item με λάθος υπολογιζόμενο bounding box**



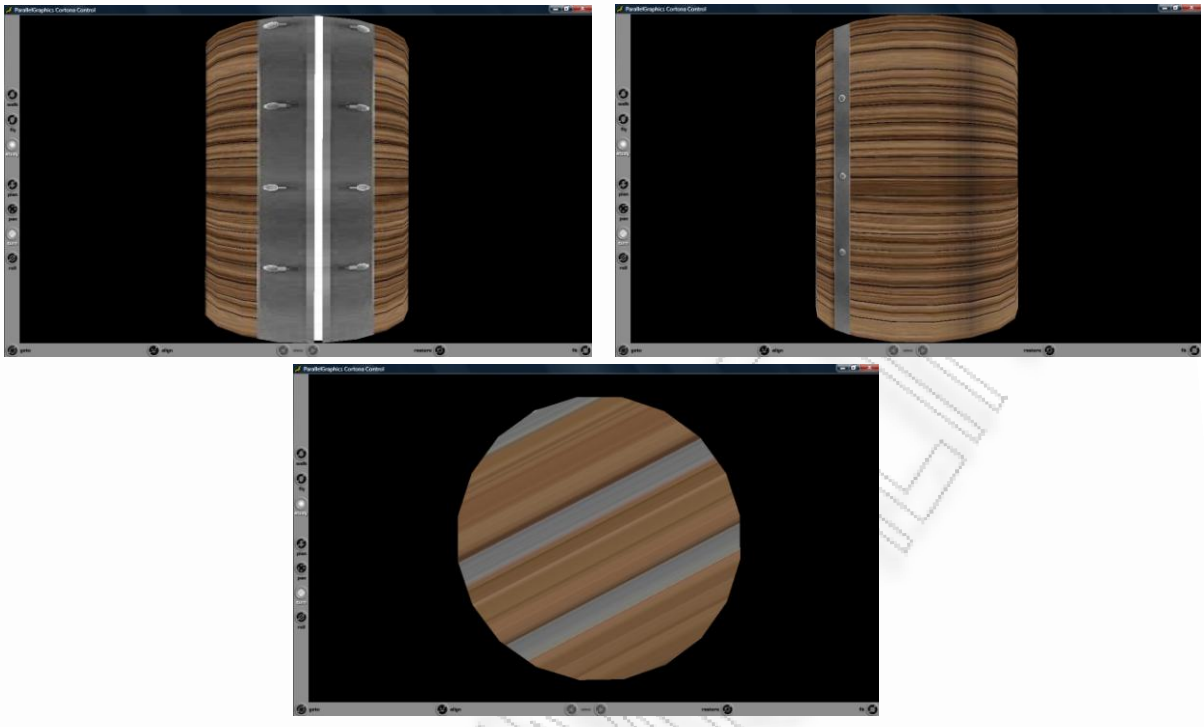

**Item με σωστό υπολογιζόμενο bounding box**

Ανεξάρτητα με τα παραπάνω οι timers, sensors και Interpolators χρησιμοποιούνται ώστε να καταστούν δυνατές οι αλληλεπιδράσεις με τον χρήστη κάτι που δεν είναι στα πλαίσια της παρούσας μεταπτυχιακής διατριβής καθώς όλες αυτές οι ενέργειες αντί να γίνονται από κάποιον χρήστη θα γίνονται από ένα ευφυή πράκτορα και οι αλληλεπιδράσεις στον κόσμο από μία ενέργεια θα ελέγχονται από έναν άλλον ευφυή πράκτορα. Έτσι οι κόμβοι αυτοί είναι περιττοί και θα πρέπει να αφαιρεθούν από την VRML. Οι μόνοι timeSensors και ColorInterpolators οι οποίοι χρησιμοποιήθηκαν τελικά είναι αυτοί της αλλαγής χρώματος κάποιων items οι οποίοι δεν προσδίδουν κάποια πληροφορία στους πράκτορες και απλά χρησιμοποιήθηκαν για να αποδώσουν περισσότερο ρεαλισμό στον κόσμο.

Οι παραπάνω όμως ενέργειες παρόλο που έλυσαν το πρόβλημα του υπολογισμού των bounding boxes δημιούργησαν ένα άλλο, αυτό της παραμόρφωσης των textures των αντικειμένων. Παραδείγματα αυτής της παραμόρφωσης δίνονται στις παρακάτω εικόνες όπου κάθε εικόνα αντιστοιχεί σε διαφορετική πλευρά του ίδιου αντικειμένου.



**Παραμόρφωση texture αντικειμένων με αρχικό geometry τύπου box**



**Παραμόρφωση texture αντικειμένων με αρχικό geometry τύπου cylinder**

Η παραμόρφωση αυτή είναι φυσική επίπτωση του tessellation καθώς έχουμε αλλάξει την διάταξη των κορυφών, των faces κ.τ.λ.. Για την επίλυση αυτού του προβλήματος προτάθηκαν δύο τεχνικές, ανάλογα με το αρχικό geometry type του κάθε item. Έτσι για geometries τύπου box χρησιμοποιήθηκε η τεχνική του textureCoordinate και για geometries τύπου cylinder, sphere και cone χρησιμοποιήθηκε η τεχνική του textureTransform.

Ένα texture ορίζεται σε ένα δισδιάστατο σύστημα συντεταγμένων (s,t) με εύρος από ο έως 1 και στις δύο διαστάσεις. Ο κόμβος textureCoordinate δέχεται ένα σύνολο σημείων του δισδιάστατου κόσμου για να καθορίσει τον τρόπο με τον οποίο εφαρμόζεται ένα texture σε ένα IndexedFaceSet. Εάν δεν έχει οριστεί τότε (όπως δηλαδή και στην περίπτωση μας) τότε το texture εφαρμόζεται στο γεωμετρικό μας αντικείμενο ως σύνολο. Ορίζοντας όμως τον κόμβο αναγκάζουμε το texture να εφαρμοστεί σε κάθε face του σχήματος ανάλογα με τις συντεταγμένες που ορίζουμε στον κόμβο. Η σύνταξη του κόμβου είναι η ακόλουθη:

```
texCoord TextureCoordinate {
                        point [ ]
                        }
texCoordIndex [ ]
```

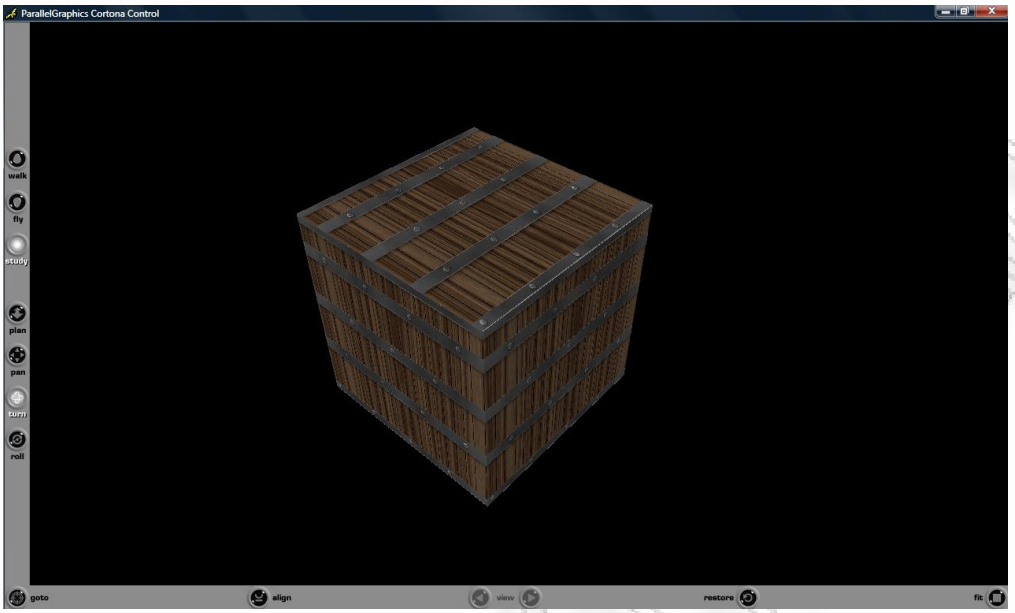
Ο κόμβος point δέχεται ως όρισμα το σύνολο των δυσδιάστατων συντεταγμένων οι οποίες καθορίζουν την επιλεγμένη περιοχή. Η σειρά των σημείων είναι σχετική και θα πρέπει να ορίζεται δεξιόστροφα με ξεκινώντας με την από την αρχή ώστε να διατηρηθεί ο προσανατολισμός του texture. Εάν το σημείο εκκίνησης είναι διαφορετικό τότε το texture περιστρέφεται κατά πολλαπλάσια των 90$^{o}$. Η παρουσίαση των σημείων σε μία δεξιόστροφη κατεύθυνση θα έχει ως αποτέλεσμα την αναπαράσταση του texture. Πιο συγκεκριμένα για ένα IndexedFaceSet το σύνολο των σημείων θα πρέπει να ταιριάζει με τον αριθμό των συντεταγμένων οι οποίες χρησιμοποιούνται για τον ορισμό ενός face, ενώ αν χρησιμοποιούνται πολλαπλά faces τότε ο αριθμός των σημείων θα πρέπει να συμφωνούν με τον αριθμό των σημείων τα οποία χρησιμοποιήθηκαν για τον ορισμό του κάθε face. Ο αριθμός των σημείων στον κόμβο θα πρέπει να είναι τουλάχιστον ίσος με τον αριθμό των σημείων του face με τον μεγαλύτερο αριθμό σημείων. Εάν ο

κόμβος texCoordIndex έχει τιμές τότε οι τιμές αυτού του πεδίου καθορίζουν τους δείκτες των points του TextureCoordinate οι οποίοι χρησιμοποιούνται για να καθορίσουν την επιλογή του κάθε face. Αντίθετα εάν ο κόμβος είναι κενός τότε χρησιμοποιείται το πεδίο coordIndex του IndexedFaceSet για να καθορίσει τους δείκτες. Στη δική μας περίπτωση έχουν οριστεί τιμές στον κόμβο texCoordIndex και οι γεωμετρίες έχουν μετατραπεί σύμφωνα με το ακόλουθο παράδειγμα:

```
geometry IndexedFaceSet {
        solid TRUE
        ccw TRUE
        colorPerVertex FALSE
        creaseAngle 1
        coord Coordinate {
                point [
                        .5 .5 .5, .5 .5 -.5,  .5
                        -.5 .5, .5 -.5 -.5, -.5 .5 .5,
                        -.5 .5     -.5,.5 -.5 .5,  -.5
                        -.5 -.5
                ]
        }
        coordIndex [
                4 0 1 5 -1
                7 3 2 6 -1
                6 2 0 4 -1
                2 3 1 0 -1
                3 7 5 1 -1
                7 6 4 5
        ]
}
```

```
geometry IndexedFaceSet {
        solid TRUE
        ccw TRUE
        colorPerVertex FALSE
        creaseAngle 1
        coord Coordinate {
                point [
                        .5 .5 .5, .5 .5 -.5, .5 -.5
                        .5, .5 -5 -.5, -.5 .5 .5, -.5
                        .5 -.5,-.5 -.5 .5, -.5 -.5 -
                        .5
                ]
        }
        coordIndex [
                4 0 1 5 -1
                7 3 2 6 -1
                6 2 0 4 -1
                2 3 1 0 -1
                3 7 5 1 -1
                7 6 4 5
        ]

        texCoord TextureCoordinate {
                point [ 1 0 0 0 1 1 0 1]
        }
        texCoordIndex [  0 1 3 2 -1 0 1 3 2 -1 0 1
                         3 2 -1 0 1 3 2 -1 0 1 3 2
                         -1 0 1 3 2 -1 ]
        }
}
```

**Πηγαίος κώδικας διαχείρισης texture geometries αρχικού τύπου box**




**Item αρχικού geometry box με διορθωμένο texture**

Όπως προαναφέρθηκε για geometries τύπου cylinder, sphere και cone χρησιμοποιήθηκε η τεχνική του textureTransform. Αυτός ο κόμβος προσαρμόζεται μέσα στον κόμβο Appearance και επιτρέπει απλούς γεωμετρικούς μετασχηματισμούς στις συντεταγμένες των textures όπως είναι αλλαγή κλίμακας, περιστροφή και μετατόπιση. Η σύνταξη του κόμβου είναι η ακόλουθη:

```
textureTransform TextureTransform {
                        scale 1 1
                        rotation 0
                        center 0 0
                        translation 0 0
}
```

Ιδιαίτερα σημαντική είναι η τιμή του πεδίου center καθώς καθορίζει το κεντρικό σημείο από το οποίο θα ξεκινήσει η αλλαγή κλίμακας και η περιστροφή. Όταν χρησιμοποιούνται όλα τα πεδία σε συνδυασμό τότε τα textures προσαρμόζονται ως εξής: πρώτα γίνεται αλλαγή κλίμακας και στην συνέχεια περιστρέφονται με βάση το center και τέλος μετατοπίζονται.

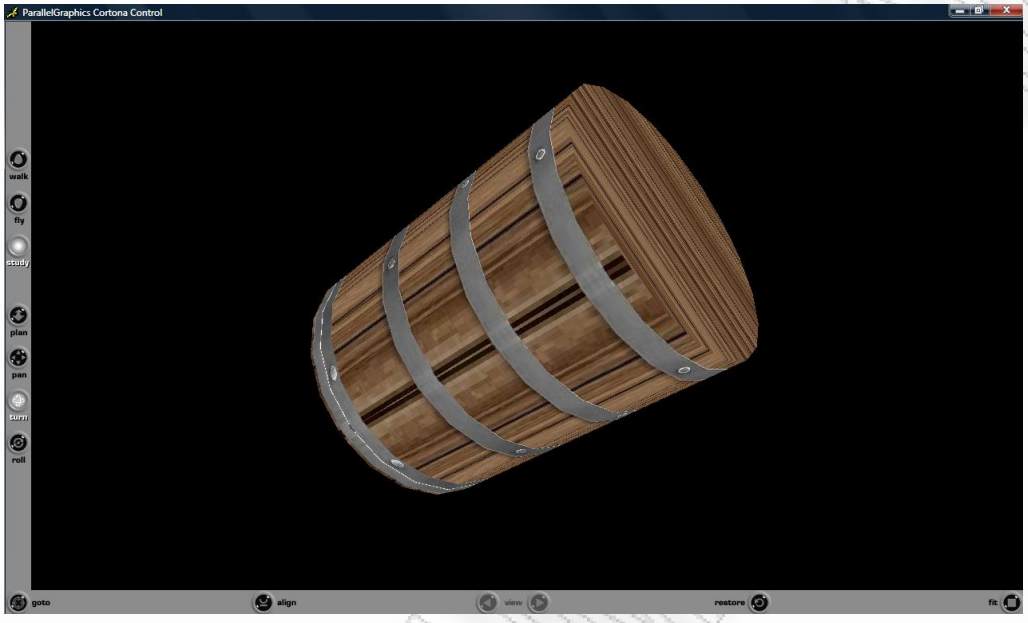
Σύμφωνα με τα παραπάνω οι αλλαγές που χρειάστηκε να γίνουν στα gevmetries μας φαίνονται στο παρακάτω παράδειγμα:

```
appearance Appearance {
        material Material {
                diffuseColor 0 0 0
        }
        texture ImageTexture { url "barrell.jpg"}
}
```

```
appearance Appearance {
        material Material {
                diffuseColor 0 0 0
        }
        texture ImageTexture { url "barrell.jpg"}
        textureTransform TextureTransform {
                scale 1 0.98
                rotation 1.57
                center 2.5 2.5
```

```
                translation 0 0
            }
        }
```

**Πηγαίος κώδικας διαχείρισης texture geometries αρχικού τύπου cylinder, sphere ή cone**




**Item αρχικού geometry cylinder με διορθωμένο texture**

Σε αυτό το σημείο έχουμε είμαστε σε θέση να ορίσουμε όλα τα items του κόσμου με virtual model representation και σωστό υπολογισμό όλων των bounding boxes, υλοποιώντας έτσι έναν πλήρη αλλά στατικό κόσμο.

## 6.2 Access Model Definition

### 6.2.1 Ορισμός

Επόμενο βήμα είναι ο ορισμός του Access Model των items για να προσθέσουμε πληροφορία σχετικά με την αναπαράσταση της λειτουργικότητας τους. Με βάση αυτή την λειτουργικότητα οι διάφοροι εικονικοί πράκτορες στον κόσμο θα είναι σε θέση να αλληλεπιδράσουν με αυτόν. Ουσιαστικά το access model για τους πράκτορες είναι ότι και οι sensors για τον χρήστη στη VRML.

Στο REVE Worlds για να ορίσουμε την λειτουργικότητα ενός item θα πρέπει να ορίσουμε ένα accesspoint. Κάθε accesspoint αντιστοιχεί σε ένα συγκεκριμένο item και επομένως οι διαστάσεις του, η κατεύθυνση και η θέση του είναι συνδεδεμένες με την δομή του κάθε item.

Για να ορίσουμε το access model ενός item, προσδιορίζουμε το item και σε αυτό ορίζουμε μια δομή access model η οποία δεν έχει attributes και μέσα σε αυτή ορίζουμε τα accesspoints τα οποία μπορεί να είναι και πάνω από ένα. Κάθε accesspoint ορίζεται από ένα όνομα “name” και από ένα κόμβο “node” ο οποίος το συνδέει με την δομή του item (μπορεί να είναι είτε ολόκληρο item είτε κόμβος του) και με αυτόν τον τρόπο αποδίδονται στο accesspoint οι φυσικές ιδιότητες οι οποίες περιγράφηκαν παραπάνω οι οποίες επειδή δεν αναθέτονται σε αυτό αλλά αποκτούνται από την συσχέτιση με το item παραμένουν και συνεχώς ενημερωμένες ανάλογα με το animation το οποίο θα εκτελέσει το item.

Το accessspoint όμως από μόνο του δεν ορίζει το είδος της λειτουργικότητας την οποία έχει το item. Για να προσδιορίσουμε αυτή την λειτουργικότητα θα πρέπει να ορίσουμε μια function. Κάθε function ορίζεται από ένα όνομα “name”, μία κλάση “class” η οποία παίρνει τιμές ανάμεσα από τρείς κλάσεις οι οποίες θα αναλυθούν παρακάτω και οι οποίες ορίζουν ακριβώς ποια θα είναι η λειτουργικότητα του

αντικειμένου, και από ένα σύνολο παραμέτρων “args” οι οποίες είναι απαραίτητες για την αρχικοποίηση της συνάρτησης.

```
<item name="door7">
        <virtualModel source="Door7">
                <transform translation="30.5 0 -10" scale="0.72 0.47 1" rotation="0 1 0 1.57"/>
        </virtualModel>
        <accessModel>
            <accessPoint name="Door7_access_p" node="Door7">
                <function
                        name="translate"
                        class="reve.scene.item.access.function.AccessPointTranslateFunctionL1"
                        args=""
                />
            </accessPoint>
        </accessModel>
</item>
```

**Ορισμός Access model**

Όπως προαναφέραμε κάθε συνάρτηση “function” του access Model μπορεί να λάβει τιμές από τρεις διαφορετικές κλάσεις, ανάλογα με το είδος της λειτουργικότητας (μετατόπιση, περιστροφή κτλ) καθώς και ανάλογα με το αν η λειτουργικότητα εφαρμόζεται σε ολόκληρο το item ή σε κάποιον κόμβο του.

Η πρώτη συνάρτηση η οποία χρησιμοποιήθηκε είναι η **AccessPointTranslateFunctionL1** η οποία χρησιμοποιείται για μετατόπιση ολόκληρων items με βάση το accesspoint τους. Για την λειτουργία της δεν είναι απαραίτητος ο ορισμός arguments για την αρχικοποίηση της συνάρτησης.

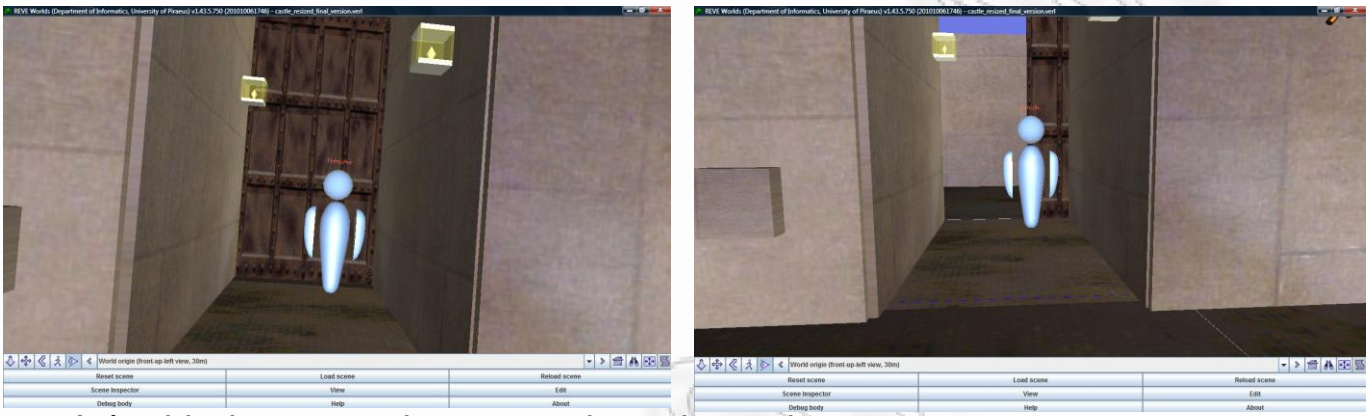
```
<item name="Door1">
        <virtualModel source="Door1" fit="true" fitCenter="true">
                <transform translation="-4.9 0 5.5" rotation="0 1 0 -1.57"/>
        </virtualModel>
        <accessModel>
            <accessPoint
                name="moverDoor"
                node="Door1">
                <function
                        name="moveDoor"
                        class="reve.scene.item.access.function.AccessPointTranslateFunctionL1"
                />
            </accessPoint>
        </accessModel>
</item>
```

**AccessPointTranslateFunctionL1**

Με τον ορισμό accesspoint και AccessPointTranslateFunctionL1 στο item Door1 κάθε εικονικός πράκτορας μπορεί να εφαρμόσει μετατόπιση στο item κάνοντας attach στο item, επιλέγοντας το accesspoint και δρώντας πάνω του.

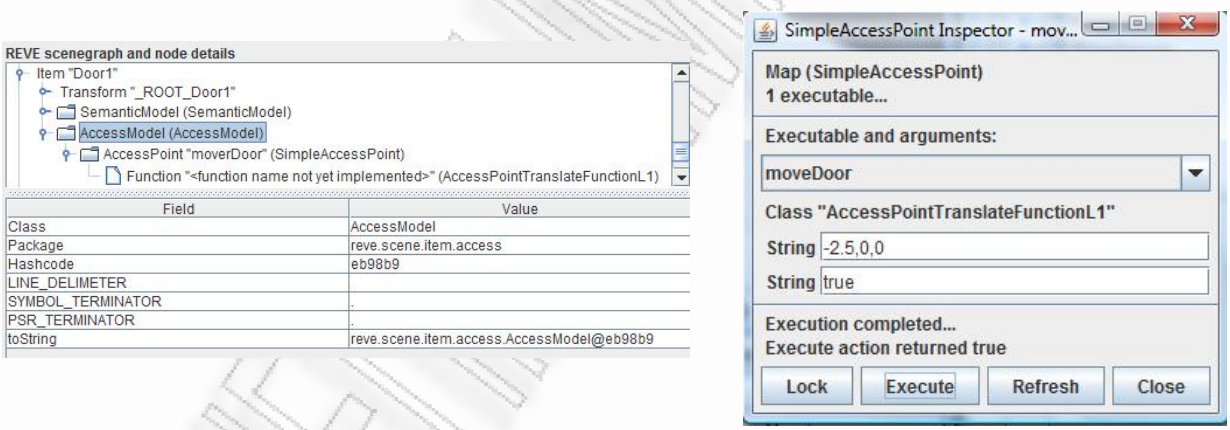
```
attach leftHand Door1 moverDoor
select leftHand move moveDoor
act leftHand move -2.5,0,0, true
```

**Έλεγχος accesspoint με AccessPointTranslateFunctionL1 από debugbot**



**Δράση debugbot σε accespoint με AccessPointTranslateFunctionL1**

Το REVE Worlds επιπρόσθετα προσφέρει την δυνατότητα του «χειροκίνητου» ελέγχου των accesspoints μέσω του SceneInspector προσφέροντας έτσι μεγαλύτερο έλεγχο και ευκολία.




**Έλεγχος των accespoint μέσω του Scene Inspector (AccessPointTranslateFunctionL1)**

Η δεύτερη συνάρτηση η οποία χρησιμοποιήθηκε είναι η **AccessPointRotateFunctionL1** η οποία χρησιμοποιείται για περιστροφή ολόκληρων items με βάση το accesspoint τους. Για την λειτουργία και αυτής δεν είναι απαραίτητος ο ορισμός arguments για την αρχικοποίηση της συνάρτησης.

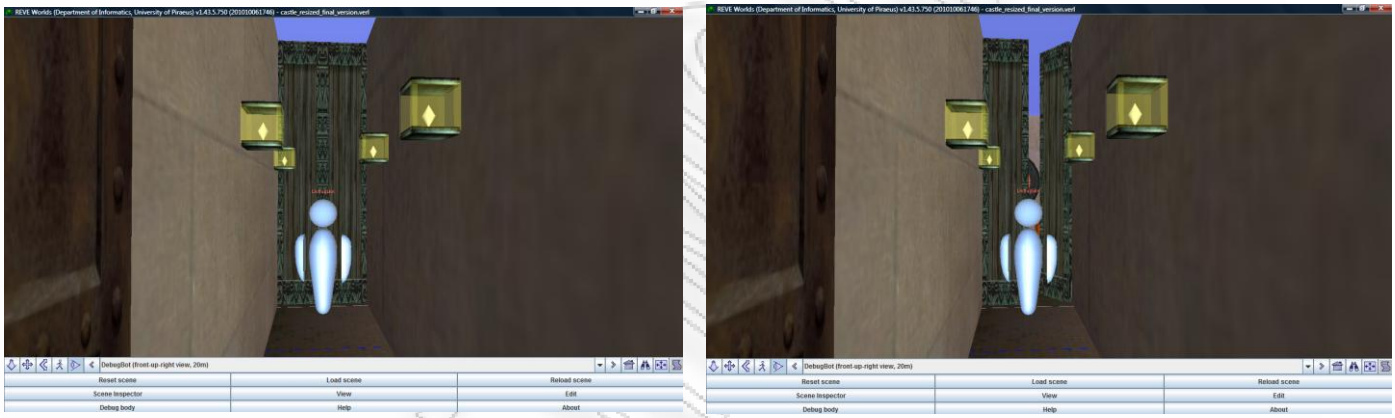
```
<item name="door4a" fit="true">
        <virtualModel source="Door4a">
                <transform translation="23.5 0 -3"/>
        </virtualModel>
        <accessModel>
            <accessPoint name="Door4_a_access_p" node="door4_acylinder">
                        <function
```

```
                    name="rotate"
                class="reve.scene.item.access.function.AccessPointRotateFunctionL1"
                    args=""
                />
            </accessPoint>
        </accessModel>
</item>
```

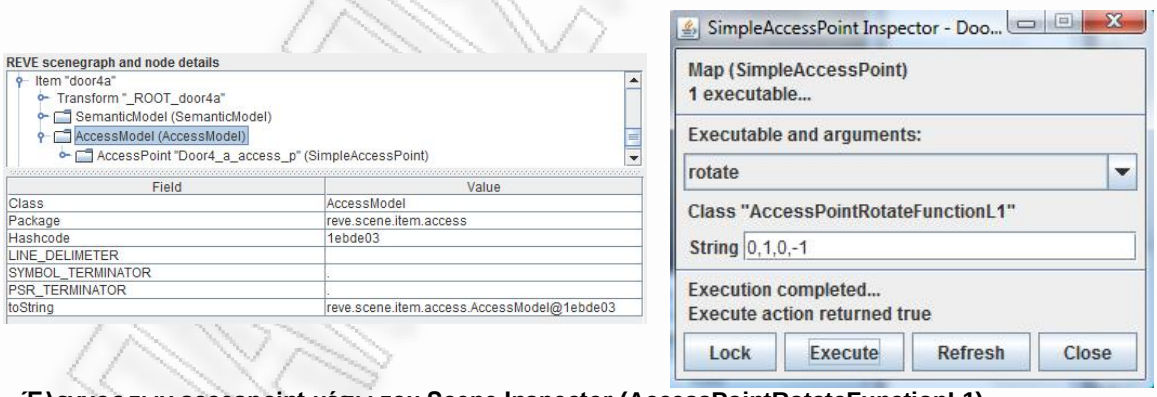
**AccessPointRotateFunctionL1**

```
attach leftHand door4a Door4_a_access_p
select leftHand turn rotate
act leftHand turn 0,1,0,1
```

**Έλεγχος accesspoint με AccessPointRotateFunctionL1 από debugbot**



**Δράση debugbot σε accespoint με AccessPointRotateFunctionL1**




**Έλεγχος των accespoint μέσω του Scene Inspector (AccessPointRotateFunctionL1)**

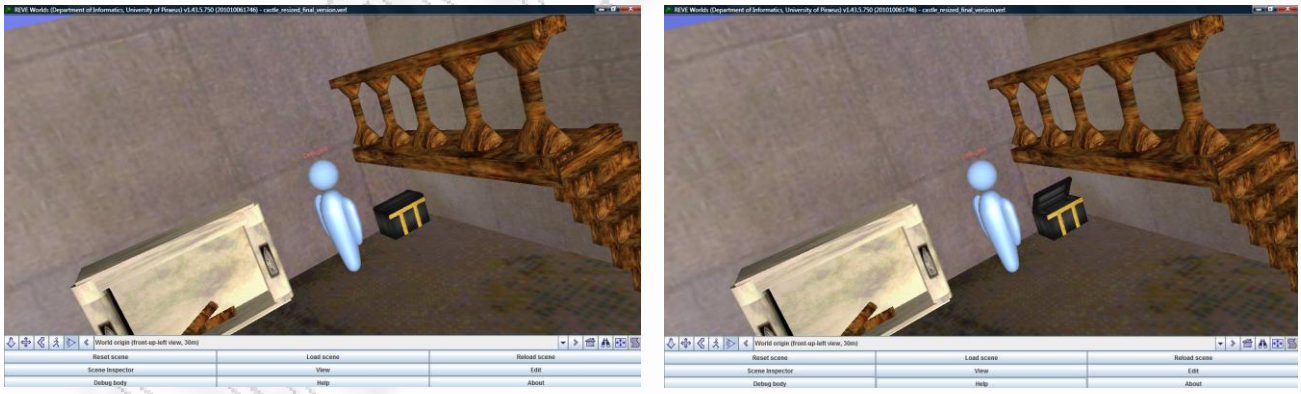
Η τρίτη και τελευταία συνάρτηση η οποία χρησιμοποιήθηκε είναι η **X3DFieldFunctionL1** η οποία χρησιμοποιείται για να μεταβάλει μια συγκεκριμένη τιμή σε ένα συγκεκριμένο πεδίο ενός συγκεκριμένου κόμβου στο virtual model, δεδομένου ότι η τιμή αυτή είναι εγγράψιμη. Η συνάρτηση αυτή χρησιμοποιήθηκε στην παρούσα μεταπτυχιακή διατριβή κυρίως για να αποδώσει λειτουργικότητα σε κόμβους των διαφόρων items. Στο πεδίο "args" δηλώνουμε κάθε φορά το είδος της λειτουργικότητας.

```
<item name="chest_room4">
        <virtualModel source="chest">
                <transform translation="4.7, 0, -11.2" rotation="0 1 0 1.57" />
        </virtualModel>
        <accessModel>
                <accessPoint name="chest_r4_open_access_p" node="top">
                        <function name="rotate"
                                class="reve.scene.item.access.function.X3DFieldFunctionL1"
                                args="rotation"
                        />
                </accessPoint>
                <accessPoint name="chest_r4_open_access_p2" node="top">
                        <function name="translate"
                                class="reve.scene.item.access.function.X3DFieldFunctionL1"
                                args="translation"
                        />
                </accessPoint>
        </accessModel>
</item>
```

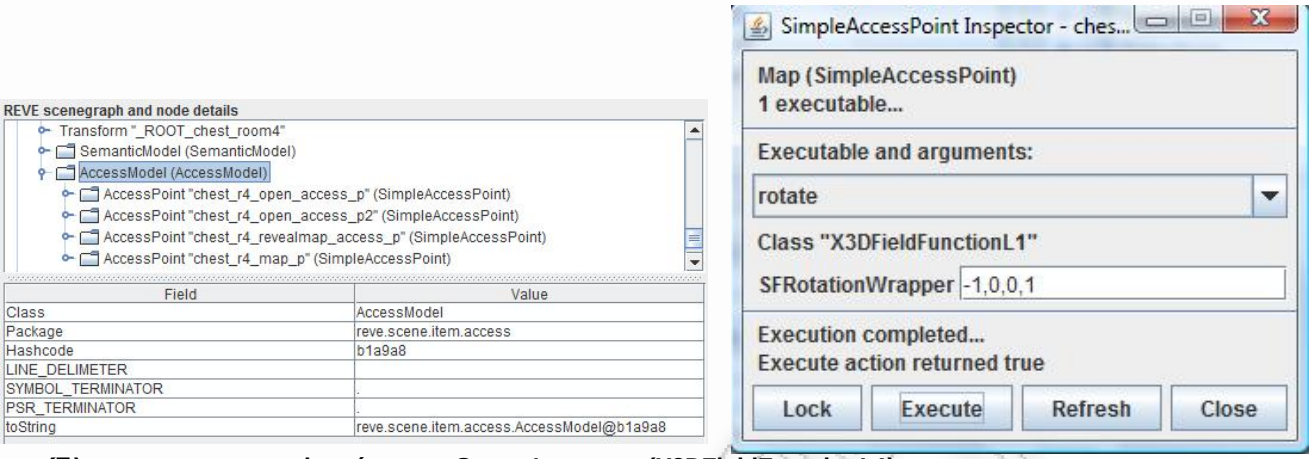
**X3DFieldFunctionL1**

```
attach leftHand chest_room4 chest_r4_revealmap_access_p
select leftHand use rotate
act leftHand use -1,0,0,1
```

**Έλεγχος accesspoint με X3DFieldFunctionL1 από debugbot**



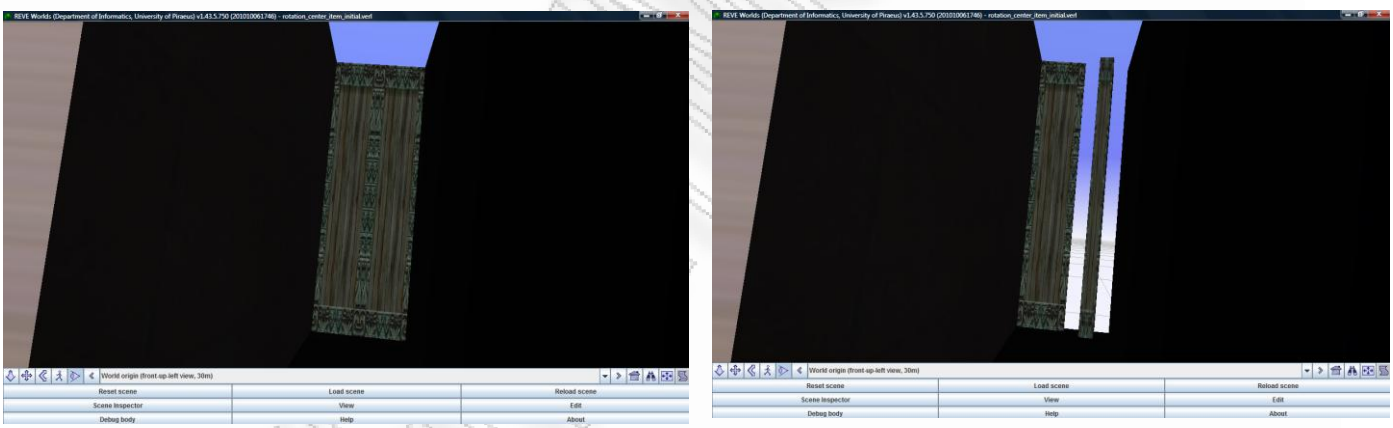
**Δράση debugbot σε accespoint με X3DFieldFunctionL1**




**Έλεγχος των accespoint μέσω του Scene Inspector (X3DFieldFunctionL1)**

### 6.2.2 Προβλήματα Εφαρμογής Functions στον κόσμο μας

Μετά τον ορισμό του access model παρατηρήθηκε ότι τα items ή οι κόμβοι των items οι οποίοι έπρεπε να περιστραφούν δεν περιστρέφονταν με βάση το κέντρο του item όπως αυτό είχε οριστεί στην VRML αλλά με βάση το κέντρο του item/κόμβου σύμφωνα με το τοπικό σύστημα συντεταγμένων. Ουσιαστικά δηλαδή ο κόμβος center από την VRML δεν κληρονομείται/περνάει στο REVE Worlds.



**Υπολογισμός του κέντρου του item από το REVE Worlds**

Για την επίλυση αυτού του προβλήματος έπρεπε να μετασχηματιστούν κατάλληλα τα αντικείμενα στη VRML έτσι ώστε με την προσθήκη ενός καινούριου κόμβου να μετατοπιστεί το κέντρο του αντικειμένου προς το σημείο το οποίο θέλουμε να είναι το καινούριο κέντρο του αντικειμένου στο REVE Worlds. Για το λόγο αυτό π.χ. για ένα item πόρτα στο σημείο προς το οποίο θέλουμε να μετατοπίσουμε το κέντρο τοποθετούμε ένα καινούριο κόμβο ο οποίος παίζει τον ρόλο ουσιαστικά των μεντεσέδων μιας πόρτας και το κέντρο του καινούριου item με δύο κόμβους παιδιά πλέον μετατοπίζεται προς αυτόν τον κόμβο προσδίδοντας έτσι ρεαλισμό στον κόσμο. Ο καινούριος αυτός κόμβος είναι ουσιαστικά κρυμμένος στην άκρη του αρχικού κόμβου ή το μέγεθος του είναι πολύ μικρό για να το αντιληφτούμε ή κρύβεται στις ενώσεις με άλλα items του κόσμου οπότε δεν δημιουργεί κάποιο αισθητικό πρόβλημα στον κόσμο μας. Πιο αναλυτικά το αρχικό item door4a έχει ένα παιδί τον κόμβο door4_aa

```
DEF Door4a Transform {
    children [
        DEF Door4_a Transform {
```

```
children[
        Transform {
                center 1.5 0 0
                scale 0.07 0.15 0.3
                translation -0.5 0 0
                children [
                        DEF door4_a  Shape {
                                appearance Appearance {
                                        material Material {
                                                diffuseColor 0 0 0
                                        }
                                texture ImageTexture {url "pue017l.jpg"}
                                }
                                geometry IndexedFaceSet {
                                        solid TRUE
                                        ccw TRUE
                                        colorPerVertex FALSE
                                        creaseAngle 1
                                        coord Coordinate {
                                                point [
                                                        7.5 20 .5, 7.5 20
                                                        -.5,  7.5  -20  .5,
                                                        7.5 -20 -.5, -7.5
                                                        20  .5,-7.5  20  –
                                                        .5, -7.5 -20 .5, -
                                                        7.5 -20 -.5
                                                ]
                                        }
                                        coordIndex [
                                                4 0 1 5 -1 7 3 2 6 -1
                                                6 2 0 4 -1 2 3 1 0 -1
                                                3 7 5 1 -1 7 6 4 5
                                        ]
                                        texCoord TextureCoordinate {
                                                point [ 1 0 0 0 1 1 0 1]
                                        }
                                        texCoordIndex [ 0 1 3 2 -1 0 1 3 2
                                                        -1 0 1 3 2 -1 0 1
                                                        3 2 -1 0 1 3 2 -1
                                                        0 1 3 2 -1 ]
                                }
                        }
                ]
        }
]
```

```
                }
        ]
}
```

**Αρχικό item Door**

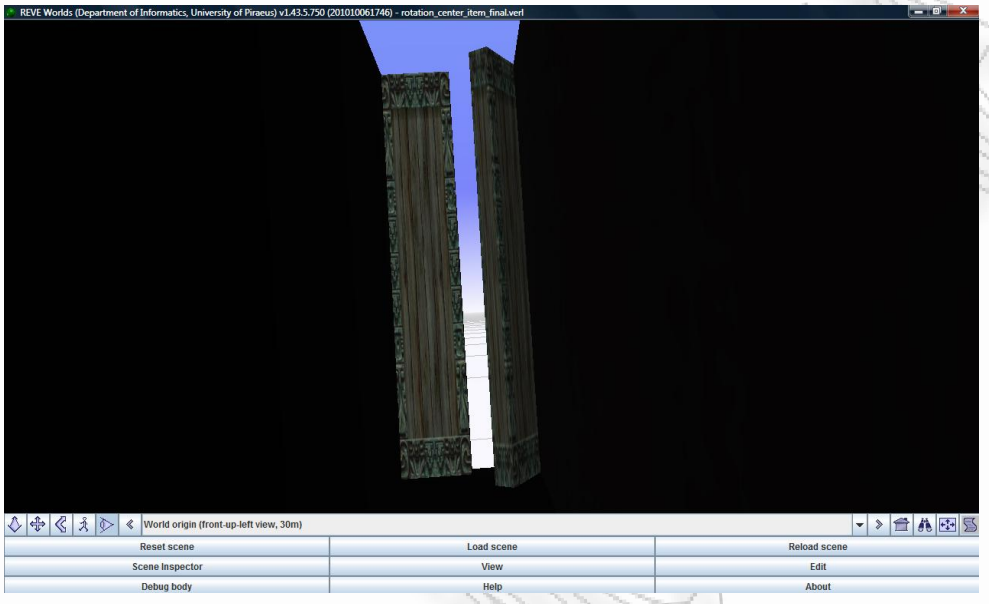
Για να μεταφέρουμε το κέντρο του παραπάνω αντικειμένου χρειάζεται να προσθέσουμε έναν ακόμη παιδί, τον κόμβο door4_acylinder

```
DEF Door4a Transform {
        children [
                DEF Door4_a Transform {
                        children[
                                Transform {
                                        scale 0.07 0.15 0.3
                                        translation -0.5 0 0
                                        children [
                                                DEF door4_aa Shape {
                                                ……..
                                                ……..
                                                ……..
                                                }
                                        ]
                                }
                        ]
                }
                DEF     door4_acylinder Shape {
                        appearance Appearance {
                                material Material {
                                        diffuseColor 0 0 0
                                }
                                texture ImageTexture { url "pue017l.jpg"}
                        }
                        geometry IndexedFaceSet {
                                solid TRUE
                                ccw TRUE
                                colorPerVertex FALSE
                                creaseAngle 1
                                coord Coordinate {
                                        point [
                                                .1 -2.9 0, .0951057 -2.9 .0309017, .0809017 -2.9
                                                .0587785,.0587785 -2.9 .0809017, .0309017 -2.9
                                                .0951056, 3.26795e-8 -2.9 .1, -.0309017 -2.9
                                                .0951057, -.0587785 -2.9 .0809017, -.0809017 -2.9
                                                .0587786, -.0951056 -2.9 .0309018, -.1 -2.9
                                                6.5359e-8, -.0951057 -2.9 -.0309016, -.0809017 –
```

```
                                    2.9 -.0587785, -.0587786 -2.9 -.0809016, -.0309018
                                     -2.9 -.0951056, -9.80385e-8 -2.9 -.1, .0309016 -2.9
                                    -.0951057, .0587784 -2.9 -.0809018, .0809016 -2.9
                                    -.0587786, .0951056 -2.9 -.0309018, .1 2.9 0,
                                    .0951057 2.9 .0309017, .0809017 2.9 .0587785
                                    .0587785 2.9 .0809017, .0309017 2.9 .0951056,
                                    3.26795e-8 2.9 .1, -.0309017 2.9 .0951057, -
                                    .0587785 2.9 .0809017, -.0809017 2.9 .0587786, -
                                    .0951056 2.9 .0309018, -.1 2.9 6.5359e-8, -
                                    .0951057 2.9 -.0309016, -.0809017 2.9 -.0587785, -
                                    .0587786 2.9 -.0809016, -.0309018 2.9 -.0951056, -
                                    9.80385e-8 2.9 -.1, .0309016 2.9 -.0951057,
                                    .0587784 2.9 -.0809018, .0809016 2.9 -.0587786,
                                    .0951056 2.9 -.0309018, 0 0 0
                            ]
                    }
                    coordIndex [
                            0 19 39 20 -1
                            1 0 20 21 -1
                            2 1 21 22 -1
                            3 2 22 23 -1
                            4 3 23 24 -1
                            5 4 24 25 -1
                            6 5 25 26 -1
                            7 6 26 27 -1
                            8 7 27 28 -1
                            9 8 28 29 -1
                            10 9 29 30 -1
                            11 10 30 31 -1
                            12 11 31 32 -1
                            13 12 32 33 -1
                            14 13 33 34 -1
                            15 14 34 35 -1
                            16 15 35 36 -1
                            17 16 36 37 -1
                            18 17 37 38 -1
                            19 18 38 39 -1
                            39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21
                            20 -1
                            0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
                    ]
                }
            }
        ]
}
```

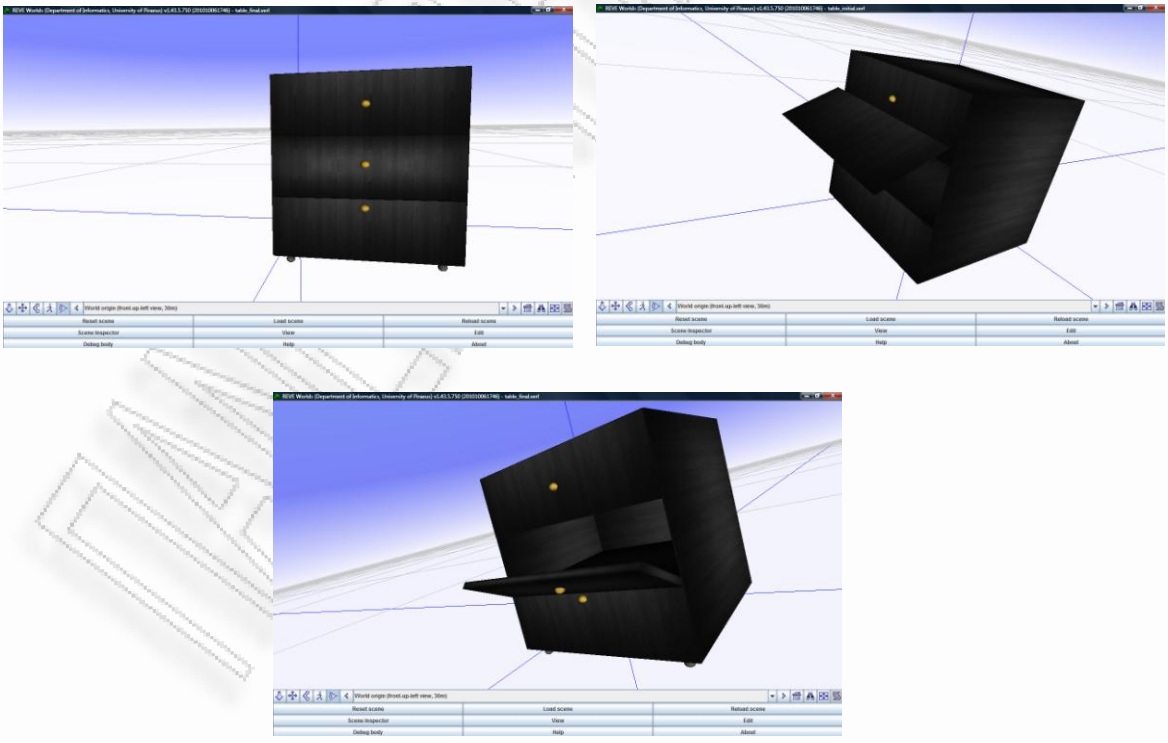
**Τελικό item Door**

Με αυτή την διόρθωση πλέον η μετατόπιση γίνεται με βάση το καινούριο κέντρο:




**Περιστροφή ως προς το καινούριο κέντρο**

Με τον ίδιο ακριβώς τρόπο αντιμετωπίζουμε και κόμβους των items που αντιμετωπίζουν το ίδιο πρόβλημα, προσθέτοντας ένα επιπλέον παιδί στο κόμβο/παιδί του item/node.



**Περιστροφή κόμβου item ως προς το αρχικό και το τελικό κέντρο**

Τέλος μια βασική διαφορά μεταξύ των AccessPointTranslateFunctionL1 και AccessPointRotateFunctionL1 με την X3DFieldFunctionL1 η οποία όμως μπορεί να προκαλέσει πρόβλημα ρεαλισμού στον κόσμο μας είναι ότι οι δύο πρώτες κατά την διάρκεια του animation μετατοπίζουν ή περιστρέφουν αντίστοιχα το item από την αρχική του θέση στην τελική σταδιακά, ενώ η τελευταία μετατοπίζει ή περιστρέφει τον κόμβο του item κατευθείαν στην τελική του θέση. Για την επίλυση αυτού του προβλήματος μπορούμε να εφαρμόσουμε πολλές διαδοχικές μικρές μετατοπίσεις ή περιστροφές από την αρχική στην τελική θέση έτσι ώστε να δημιουργείται το εφέ της σταδιακής κίνησης.

## 6.3 Semantic Model Definition

### 6.3.1 Ορισμός

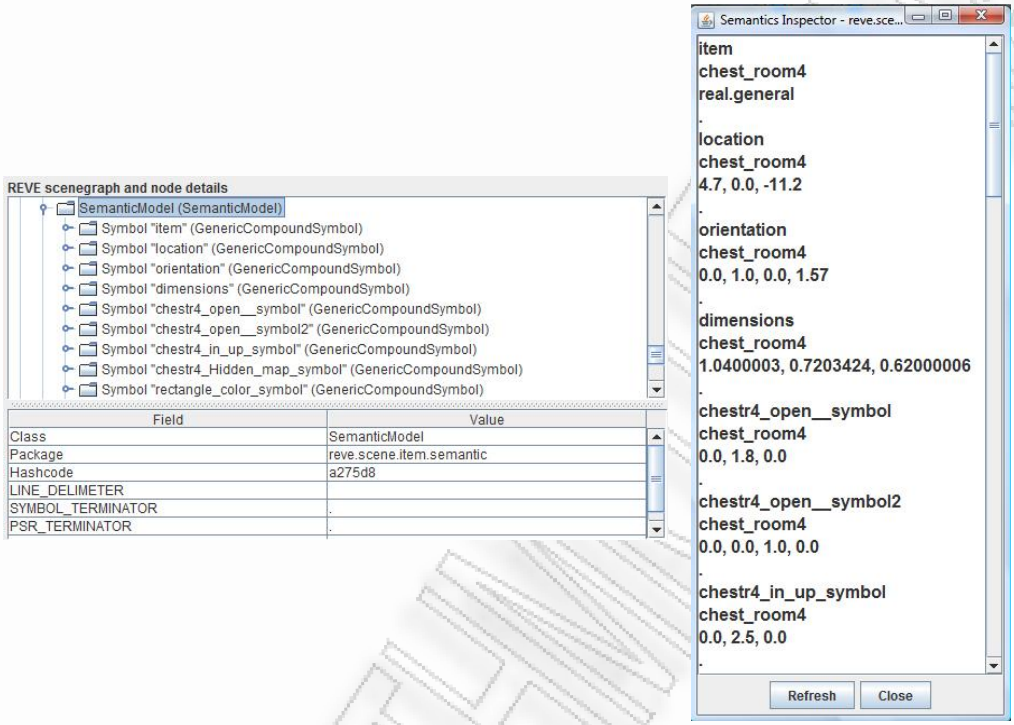
Το τελευταίο βήμα για την ολοκλήρωση της αναπαράστασης του κόσμου είναι ο ορισμός του semantic model, η αναπαράσταση δηλαδή της πληροφορίας η οποία είναι διαθέσιμη προς αντίληψη. Η διαθέσιμη αυτή πληροφορία αναπαριστάται με σύμβολα κάθε ένα από τα οποία περικλείει κάθε φορά κάποια συγκεκριμένη πληροφορία για το item. Κάθε item μπορεί να περιλαμβάνει απεριόριστο αριθμό συμβόλων.

Για να ορίσουμε το semantic model, προσδιορίζουμε το item στο οποίο θέλουμε να προσθέσουμε πληροφορία και προσθέτουμε σε αυτό έναν κόμβο semanticModel ο οποίος δεν έχει ορίσματα. Ορίζοντας έναν κόμβο semanticModel για ένα item, το σύστημα υπολογίζει για αυτό τέσσερα σύμβολα, με πληροφορία σχετικά με το item και την κλάση στην οποία ανήκει, τη θέση του, τον προσανατολισμό του και το μέγεθος του bounding box που το περικλείει. Επόμενο βήμα είναι ο ορισμός συμβόλων πέρα από αυτά που ορίζονται by default τα οποία αναπαριστούν πρόσθετη πληροφορία την οποία χρειαζόμαστε. Για να ορίσουμε ένα επιπλέον σύμβολο για ένα item προσθέτουμε έναν κόμβο symbol μέσα στο semanticModel του item και ορίζουμε ένα όνομα name για αυτό. Μπορούμε προαιρετικά σε αυτό το σημείο να ορίσουμε έναν κόμβο node στον οποίο αντιστοιχεί το σύμβολο και ο οποίος είναι είτε είναι ολόκληρο το item είτε είναι κόμβος του item. Στη συνέχεια πρέπει να ορίσουμε τον κόμβο arguments του συμβόλου, οποίος αποτελείται από τον προσδιορισμό της κλάσης η οποία θα χρησιμοποιηθεί καθώς από τα δεδομένα αρχικοποίησης “args”. Στη συγκεκριμένη μεταπτυχιακή διατριβή έχει χρησιμοποιηθεί μία μόνο κλάση από τις δύο συνολικά τις οποίες παρέχει η αναπαράσταση REVE, η X3DConnectorL1. Στα δεδομένα αρχικοποίησης της “args” ορίζουμε το είδος της πληροφορίας την οποία θέλουμε να αναπαραστήσουμε και σε περίπτωση όπου δεν έχουμε ορίσει προηγουμένως τον κόμβο node ορίζουμε σε αυτό το σημείο τον κόμβο ο οποίος αντιστοιχεί στο virtual model και στον οποίο αποδίδουμε την πληροφορία. Κάθε φορά που υπάρχει κάποια μεταβολή στην τιμή του συμβόλου τότε η τιμή του ενημερώνεται αυτόματα.

```
<item name="table5">
        <virtualModel source="table5">
                <transform translation="17 0 -11.9"/>
        </virtualModel>
        <semanticModel>
                <symbol name="table5_symbol_translation" node="t17a">
                        <argument
                                class="reve.scene.item.semantic.argument.X3DConnectorL1"
                                args="t17a, translation"
                        />
                </symbol>
        </semanticModel>
</item>
```

**Ορισμός SemanticModel**

Το REVE Worlds μας επιτρέπει για περισσότερη ευκολία να έχουμε πρόσβαση στις τιμές των συμβόλων καθώς μέσω στου sceneInspector δίνεται πρόσβαση στον semanticInspector όπου μπορούμε να δούμε συγκεντρωμένες τις τιμές όλων των συμβόλων ενός item.




**Semantic Inspector**

Όσων αφορά την παρούσα μεταπτυχιακή διατριβή ορισμός συμβόλων χρησιμοποιήθηκε ώστε να υπάρχει διαθέσιμη πληροφορία για τους πράκτορες του κόσμου σχετικά με την θέση και την μετατόπιση ή περιστροφή κόμβων των items, καθώς και πληροφορία σχετικά με ιδιότητες των items όπως παραδείγματος χάριν το χρώμα κάποιου item.

Σε αυτό το σημείο έπειτα και από τον ορισμό του semanticModel έχει ολοκληρωθεί ο δομικός προσδιορισμός του κόσμου μας, ενός κόσμου ο οποίος είναι πλήρως λειτουργικός και πέρα από ρεαλιστικός παρέχει όλες τις απαραίτητες πληροφορίες στους πράκτορες ώστε να μπορέσουν να αλληλεπιδράσουν. Ο πλήρης ορισμός ενός item σύμφωνα με τα παραπάνω απεικονίζεται αμέσως μετά.

```
<item name="rhomb_mechanism">
        <virtualModel source="rhomb_mechanism">
                <transform translation="42.5 0 -11.5"/>
        </virtualModel>
        <accessModel>
                <accessPoint
                        name="rhomb_mechanism_symbol_access_p"
                        node="romssymbol_buttons">
                                <function name="translate"
                                    class="reve.scene.item.access.function.X3DFieldFunctionL1"
                                    args="translation"
                                />
```

```
        </accessModel>
        <semanticModel>
                <symbol name="rhombmain_symbol">
                        <argument
                                class="reve.scene.item.semantic.argument.X3DConnectorL1"
                                args="rom_symbol, translation"
                        />
                </symbol>
                <symbol name="yellowrhomb_color_symbol">
                        <argument
                                class="reve.scene.item.semantic.argument.X3DConnectorL1"
                                args="yellow_rhomb_symbol, diffuseColor"
                        />
                </symbol>
        </semanticModel>
</item>
```

**Συνολικός ορισμός item**

# 7. Εικονικοί Πράκτορες

Στον κόσμο μας υπάρχουν δύο είδη πρακτόρων. Ο ένας πράκτορας είναι ένας πράκτορας ο οποίος έχει το ρόλο του «θεού». Δεν έχει σώμα, και έχει πληροφορία για οτιδήποτε συμβαίνει μέσα στον κόσμο καθώς έχει την εποπτεία του. Ουσιαστικά ο πράκτορας έχει βασιστεί στην υλοποίηση του housekeeper, μιας java βιβλιοθήκης που υποστηρίζει την ανάπτυξη αυτών των ειδικών πρακτόρων. Ο πράκτορας housekeeper έχει μόνο έναν hand effector με actions move, turn και use.

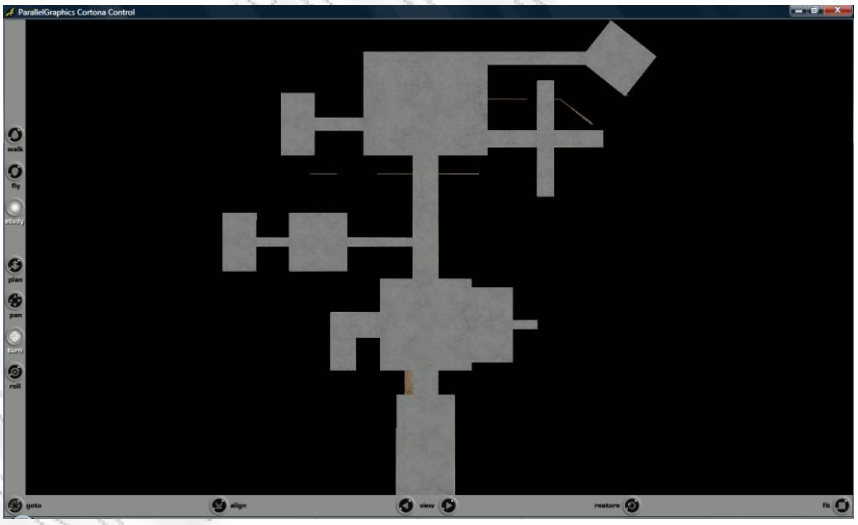
Ο δεύτερος πράκτορας που υπάρχει στον κόσμο μας είναι ο πράκτορας παίχτης, υλοποιημένος με βάση την SARA, μια βιβλιοθήκη υλοποιημένη σε JAVA η οποία επιτρέπει την ανάπτυξη πρακτόρων με σύνθετες λειτουργίες pathfinding. wandering, χωρικής αναπαράσταση, ελέγχου της βάσης γνώσης και εξαγωγής συμπερασμάτων από αυτή καθώς και άλλων πολύπλοκων διαδικασιών. Ο SARA Agent έχει leftHand και rightHand effectors με τα αντίστοιχα actions. Δεν έχει καμία γνώση του κόσμου αν τον εξερευνήσει και αντιλαμβάνεται τις αλλαγές στον κόσμο μόνο κάνοντας sense στα items του κόσμου και λαμβάνοντας πληροφορία από αυτά.

Για να είναι σε θέση το σύστημα να διαχωρίζει τους πράκτορες μεταξύ τους, κάθε πράκτορας συνδέεται σε διαφορετικό port, ο μεν housekeeper στο port 4445, ο δε SARA στο port 4444.

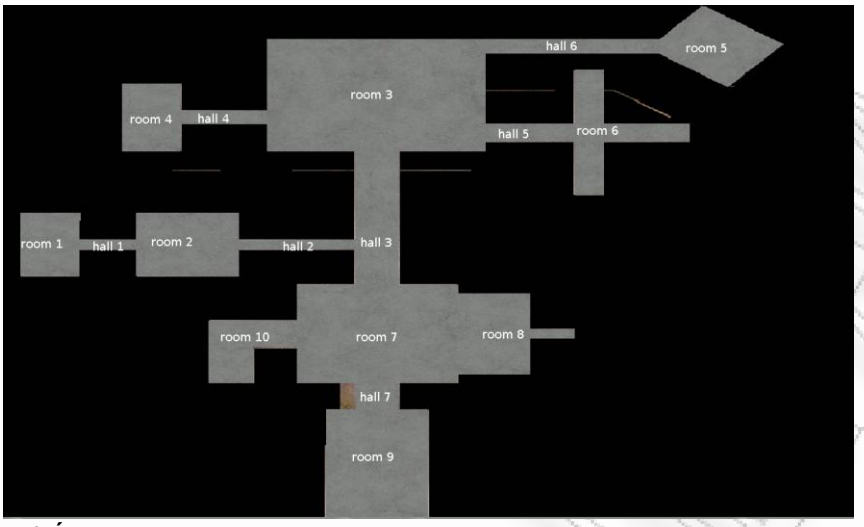
Στη συνέχεια αρχικά θα παρουσιάσουμε την συμπεριφορά την οποία θα πρέπει να έχουν οι πράκτορες μέσα στον κόσμο, δηλαδή το walkthrough του κόσμου μας και έπειτα θα αναλύσουμε πως υλοποιήθηκε αυτή η συμπεριφορά

## 7.1 Walkthrough

Ο κόσμος αποτελείται από 10 δωμάτια, τα οποία συνδέονται μεταξύ τους με βοηθητικούς διαδρόμους. Στις παρακάτω εικόνες βλέπουμε την κάτοψη του κόσμου, καθώς και την αντιστοίχηση των δωματίων με αρίθμηση η οποία μας βοηθά τόσο στην αναπαράσταση του κόσμου όσο και στην περιγραφή του.




**Κάτοψη κόσμου**

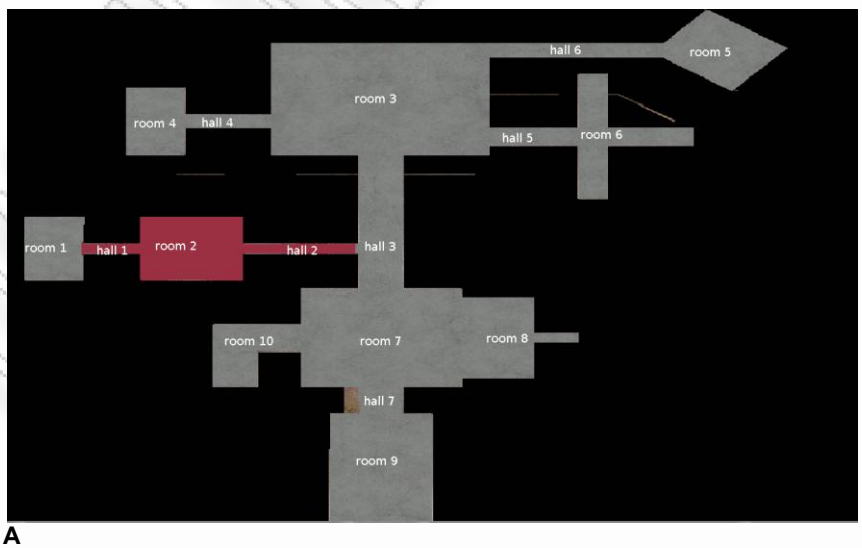



**Αρίθμηση δωματίων**

Αρχικά ο πράκτορας παίχτης (SARA Agent) φορτώνεται στο δωμάτιο 2 (room 2) και συγκεκριμένα στις συντεταγμένες 2,2 του κόσμου, με προσανατολισμό 3.14 ακτίνια. Εάν δεν ορίσουμε το σημείο όπου θα εμφανιστεί για πρώτη φορά ο πράκτορας μας τότε αυτός θα εμφανιστεί κάπου τυχαία μέσα σε αυτόν. Ο λόγος που ο πράκτορας φορτώνεται πάντα σε ένα συγκεκριμένο σημείο του κόσμου είναι για να αποφύγουμε να φορτωθεί στο εξωτερικό τμήμα του κάστρου και επομένως να μην μπορεί να μπει μέσα σε αυτό, καθώς και για να αποφύγουμε να φορτωθεί σε σημείο όπου θα εγκλωβιστεί στο εσωτερικό κάποιου bounding box κάποιου item. Για να ορίσουμε συγκεκριμένες συντεταγμένες εμφάνισης του πράκτορα ορίζουμε στο αρχείο reve.properties τις μεταβλητές:

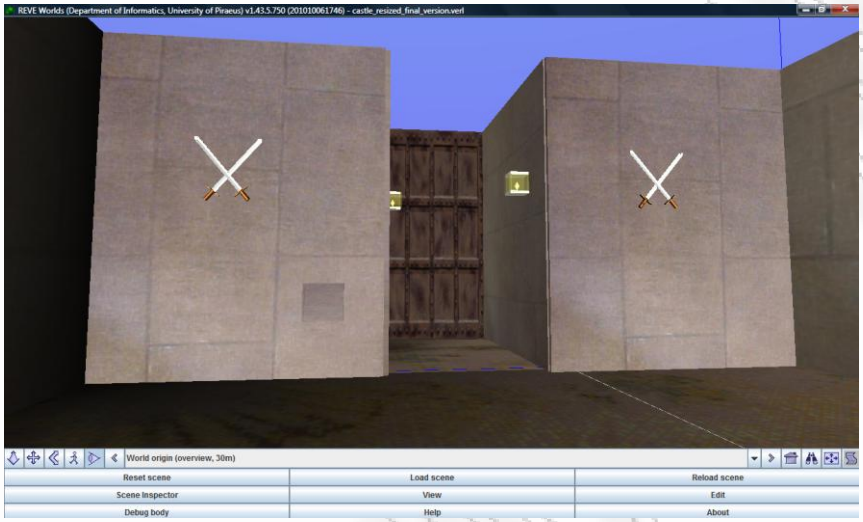
```
aip.tcpipbodycontroller.spawnlocation=2 2
aip.tcpipbodycontroller.spawnorientation=3.14
```

Από το δωμάτιο που βρίσκεται ο πράκτορας έχει πρόσβαση στους διαδρόμους 1 και 2 (hall1 και hall2) - Κατάσταση Α.

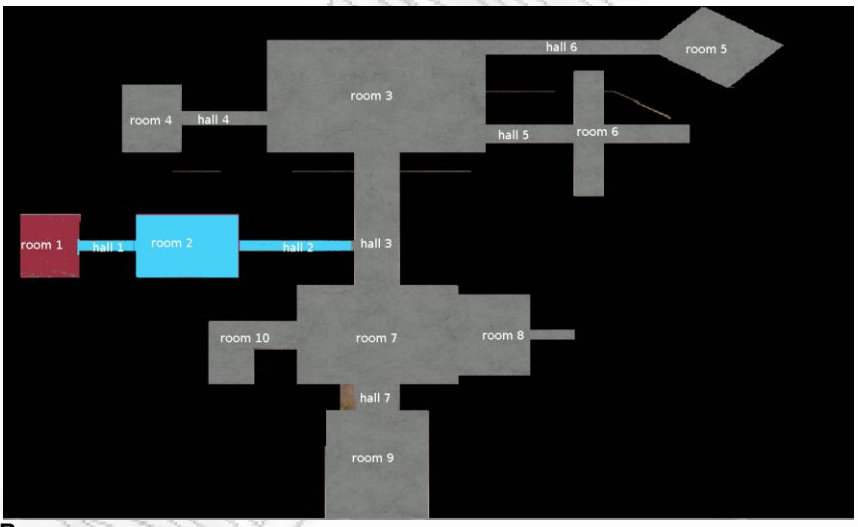



**Κατάσταση Α**

Στο δωμάτιο 2 και συγκεκριμένα πριν τον διάδρομο 1 υπάρχει ένας διακόπτης ο οποίος ανοίγει την πόρτα για το δωμάτιο 1 (room 1). Ο πράκτορας παίχτης πρέπει να πατήσει αυτόν τον διακόπτη. Μόλις ο πράκτορας θεός αντιληφθεί ότι ο πράκτορας παίχτης ανακάλυψε και άνοιξε τον διακόπτη τότε ανοίγει την πόρτα για το δωμάτιο 1 ως αποτέλεσμα του ανοίγματος του διακόπτη (Κατάσταση Β).

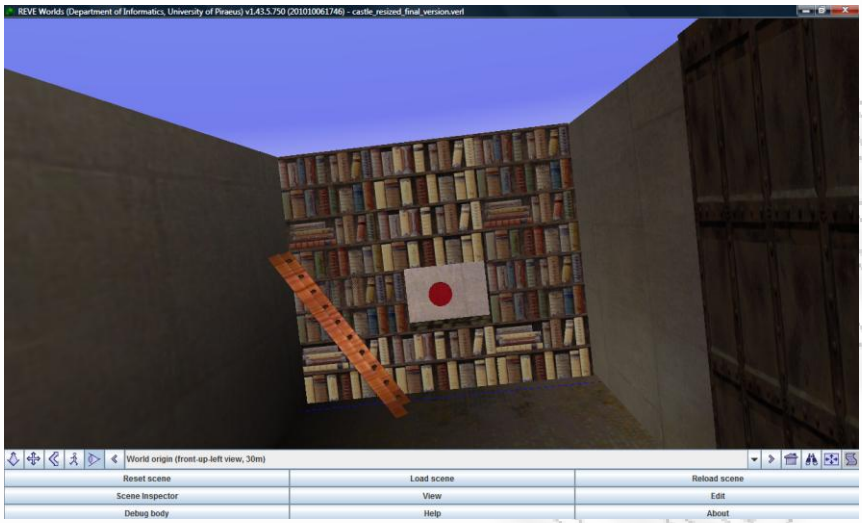



**Δωμάτιο 2 – Πρόσβαση στο Δωμάτιο 1**

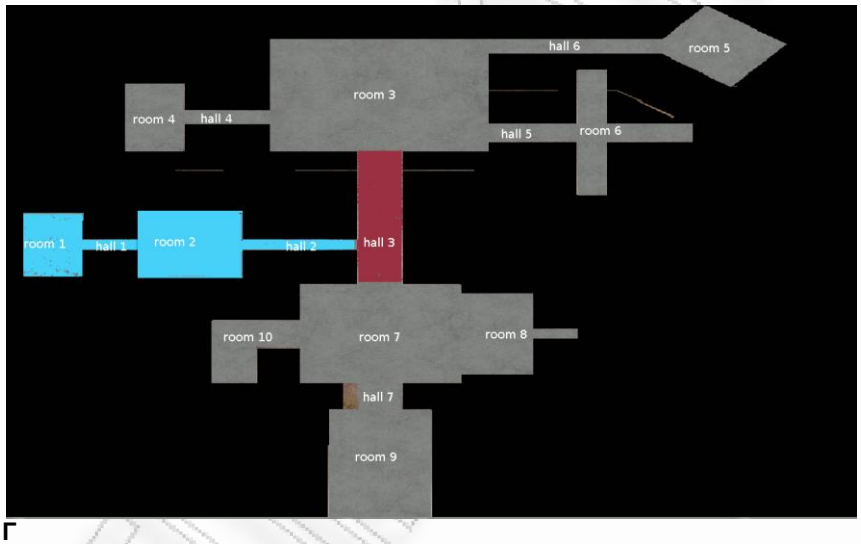



**Κατάσταση Β**

Στο δωμάτιο 1 ο πράκτορας παίχτης πρέπει να βρει με ποιον τρόπο θα ανοίξει η πόρτα που χωρίζει τον διάδρομο 2 με τον διάδρομο 3 (hall 3). Σε αυτό το δωμάτιο υπάρχει μια βιβλιοθήκη. Μόλις ο πράκτορας παίχτης κάνει sense ένα συγκεκριμένο ράφι τότε πλησιάζει κοντά σε αυτό. Ο πράκτορας housekeeper ελέγχει τη θέση του πράκτορα παίχτη και μόλις αυτός πλησιάσει αρκετά κοντά στη βιβλιοθήκη και ξεπεράσει κάποιες συγκεκριμένες συντεταγμένες τότε ανοίγει το ράφι της βιβλιοθήκης πίσω από το οποίο υπάρχει ο διακόπτης που ανοίγει την πόρτα για τον διάδρομο 3. Μόλις ο πράκτορας παίκτης αντιληφθεί αυτόν τον διακόπτης τον ανοίγει και με την σειρά του ο housekeeper μόλις ανοίξει ο διακόπτης ανοίγει την πόρτα για τον διάδρομο 3.

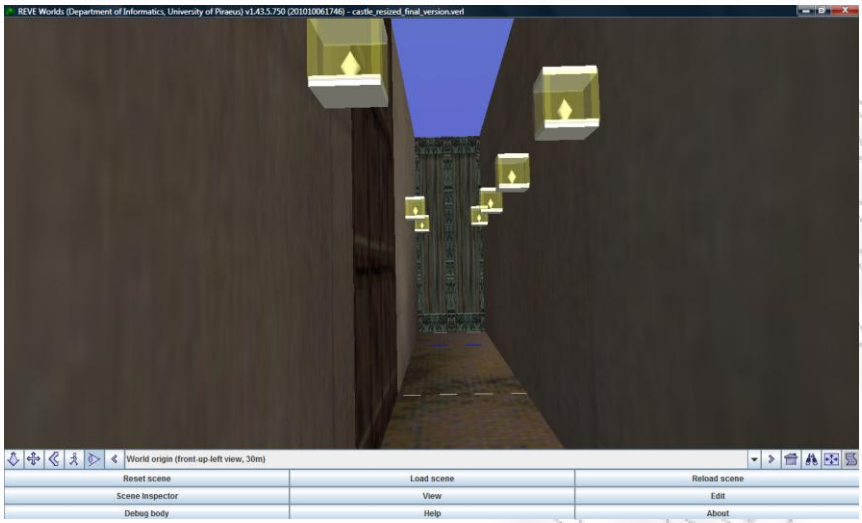



**Δωμάτιο 1 - Διακόπτης πόρτας με πρόσβαση στην κατάσταση Γ**

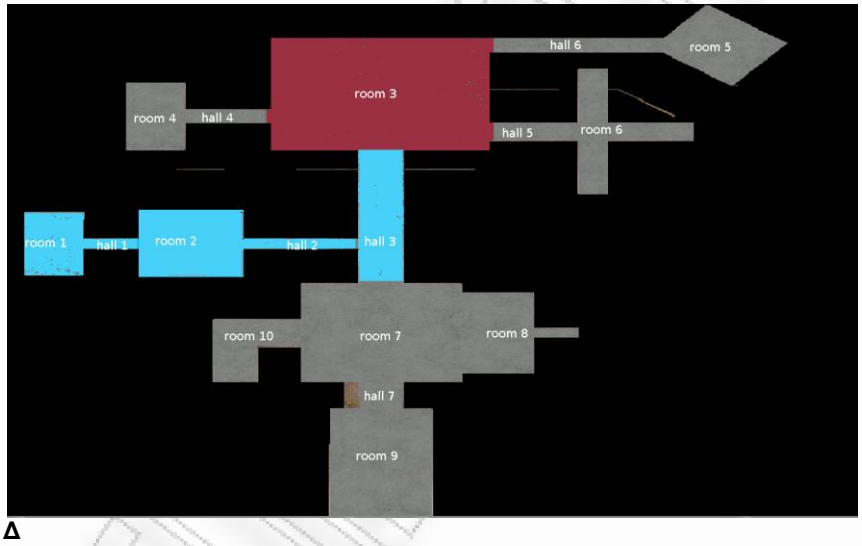



**Κατάσταση Γ**

Όταν ο πράκτορας παίχτης βρίσκεται στην κατάσταση Γ έχει πρόσβαση στον διάδρομο 3 στον οποίο βρίσκονται δύο πόρτες, όπου η μία δίνει πρόσβαση στο δωμάτιο 3 (room 3) και η άλλη δίνει πρόσβαση στο δωμάτιο 7 (room 7). Πιο συγκεκριμένα η πόρτα που δίνει πρόσβαση στο δωμάτιο 7 είναι κλειδωμένη, η πόρτα όμως που δίνει πρόσβαση στο δωμάτιο 3 είναι μεν κλειστή αλλά όχι και κλειδωμένη. Έτσι το μόνο που έχει να κάνει ο πράκτορας παίχτης είναι να «σύρει» την πόρτα και να έχει πρόσβαση στο δωμάτιο 3 (Κατάσταση Δ)




**Διάδρομος 3 – Πόρτα δωματίου 3**

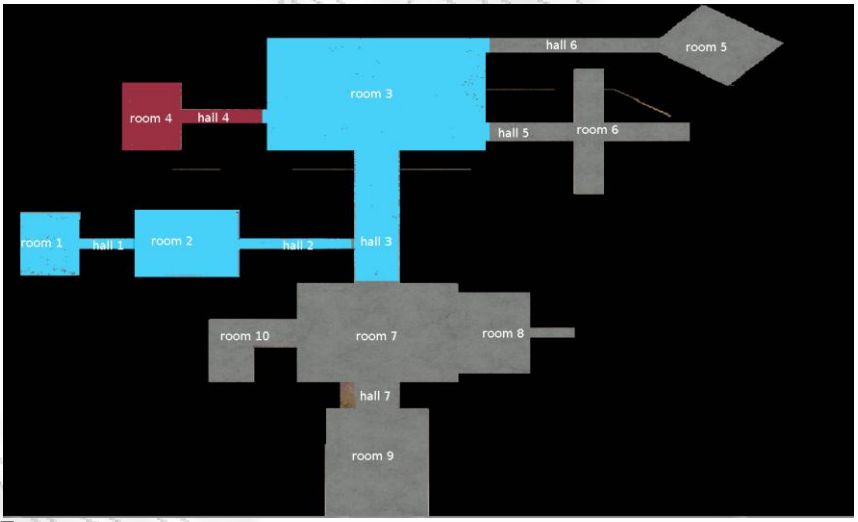



**Κατάσταση Δ**


**Δωμάτιο 3**

Από το δωμάτιο 3 ο πράκτορας παίχτης πρέπει να βρει με ποιον τρόπο θα αποκτήσει πρόσβαση στον διάδρομο 4 (hall 4) και στο δωμάτιο 4 (room 4). Για να το καταφέρει αυτό πρέπει να βρει τον κρυμμένο διακόπτη ο οποίος ενεργοποιεί την πόρτα για τον διάδρομο 4. Πιο συγκεκριμένα κάνοντας sense στο τραπέζι το οποίο βρίσκεται στον βόριο τοίχο του δωματίου, αυτό ανοίγει. Μόλις ο πράκτορας θεός αντιληφθεί ότι έχει ανοίξει το συρτάρι του τραπεζιού εμφανίζει ένα κλειδί να βγαίνει από το συρτάρι και μόλις ολοκληρωθεί η κίνηση του κλειδιού πάλι ο πράκτορας θεός ανοίγει την πόρτα για το δωμάτιο 4 (κατάσταση Ε).


**Δωμάτιο 3 – Διακόπτης πρόσβασης στο δωμάτιο 4**



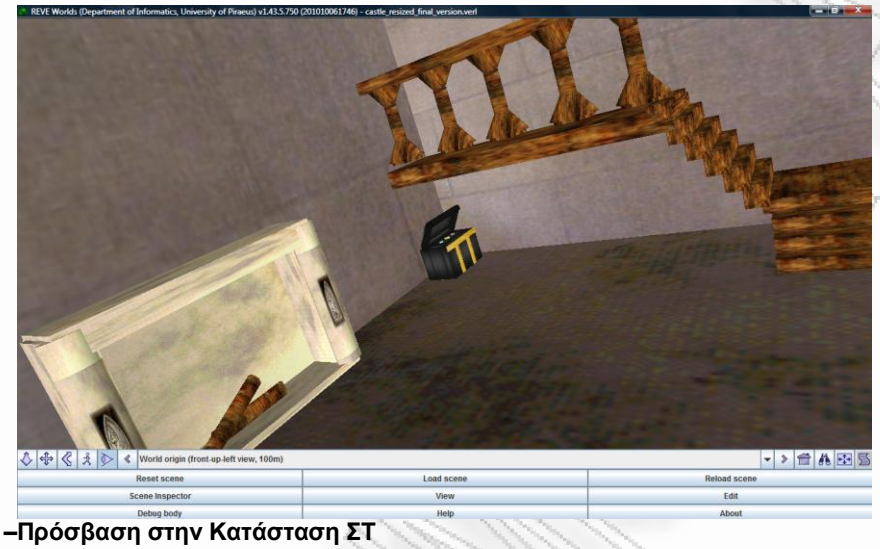

**Κατάσταση Ε**

Στην κατάσταση Ε και πιο συγκεκριμένα στο δωμάτιο 4 ο πράκτορας παίχτης πρέπει να βρει κάποιο συνδυασμό, ο οποίος θα τον βοηθήσει στη συνέχεια να επιλύσει κάποιο "puzzle" και να ανοίξει περισσότερες πόρτες στα δωμάτια του κάστρου. Επιπλέον βρίσκοντας αυτόν τον συνδυασμό – κωδικό, ανοίγουν οι πόρτες που δίνουν πρόσβαση στους ακόλουθους χώρους:

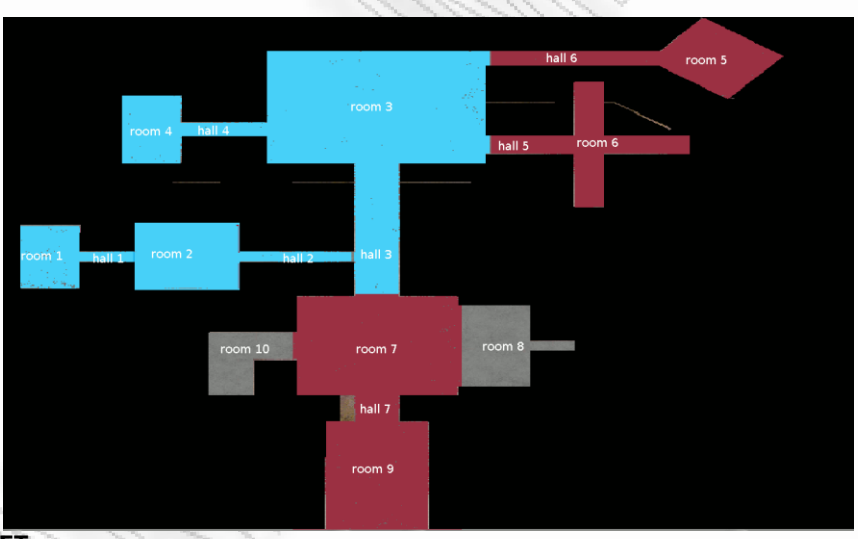
1. Διάδρομος 6 – Δωμάτιο 5 (hall 6 - room 5)
2. Διάδρομος 5 – Δωμάτιο 6 (hall 5 – room 6)
3. Δωμάτιο 7 – Διάδρομος 7 – Δωμάτιο 9 (room7 – hall 7 –room 9)

και ο πράκτορας περνάει στην κατάσταση ΣΤ

Πιο συγκεκριμένα ο πράκτορας παίχτης πρέπει να βρει μέσα στο δωμάτιο 4 ένα μπαούλο. Μόλις κάνει sense αυτό το μπαούλο ελέγχει αν είναι κλειστό. Αν είναι κλειστό το ανοίγει. Αμέσως μετά ελέγχει αν ο πάτος του μπαούλου είναι κλειστός. Αν ναι τον ανοίγει. Τότε ο πράκτορας θεός εμφανίζει μέσα από το μπαούλο έναν συνδυασμό συμβόλων με γεωμετρικά σχήματα και σύμβολα τα οποία βλέπει ο πράκτορας παίχτης. Παράλληλα ο πράκτορας θεός αφού εμφανίσει τον χάρτη/κωδικό ανοίγει και τις πόρτες για τους χώρους που προαναφέρθηκαν.

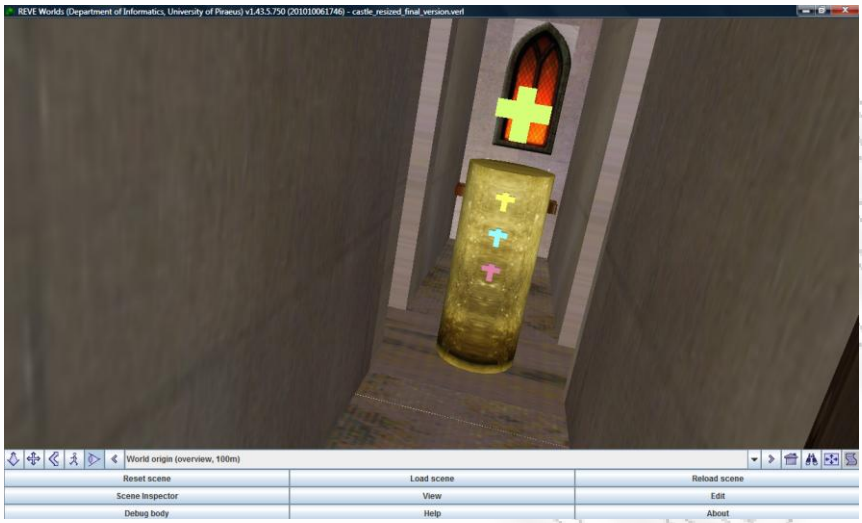



**Δωμάτιο 4 –Πρόσβαση στην Κατάσταση ΣΤ**

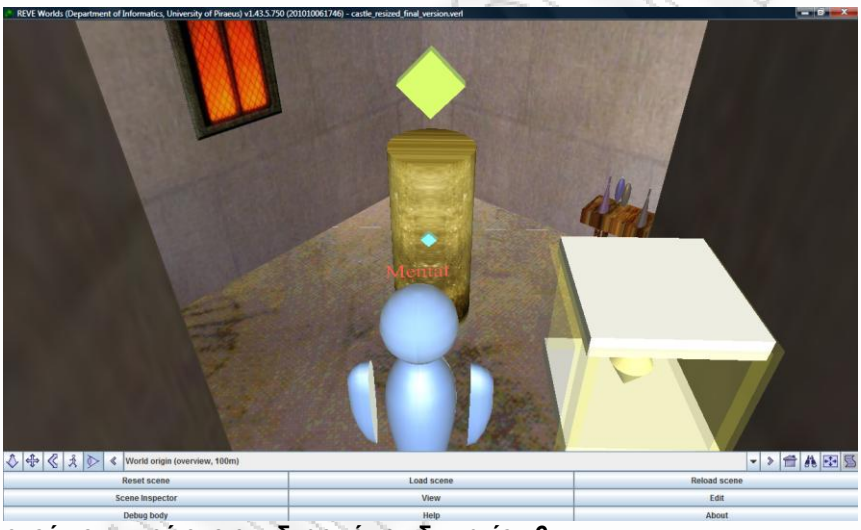



**Κατάσταση ΣΤ**

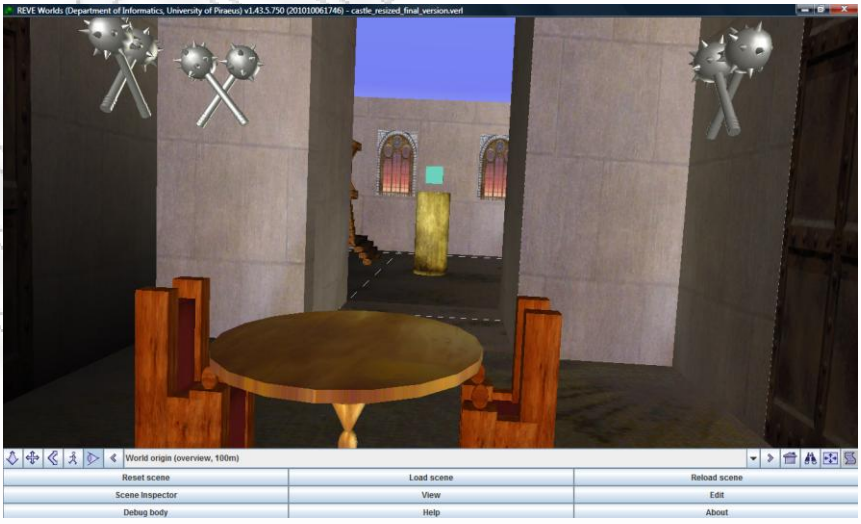
Στα δωμάτια 6, 7 και 9 βρίσκονται τρεις μηχανισμοί που αντιστοιχούν ο καθένας στα γεωμετρικά σχήματα τα οποία βρήκε ο πράκτορας παίχτης στο συνδυασμό του μπαούλου. Μόλις ο πράκτορας παίχτης κάνει sense κάθε ένα από αυτούς τους μηχανισμούς ελέγχει τη θέση του γεωμετρικού σχήματος στην κορυφή του και την μετακινεί. Μόλις μετακινηθεί αυτό το σύμβολο ο πράκτορας θεός αποκαλύπτει τρία κουμπιά με διαφορετικό σχήμα το καθένα. Μόλις ο πράκτορας παίχτης δει αυτά τα κουμπιά τα συγκρίνει χρωματικά με αυτά που είδε μέσα στο μπαούλο, «κρύβει» πάλι πίσω στο μηχανισμό τα λανθασμένα και μετακινεί προς τα μπροστά το σωστό κουμπί. Ο πράκτορας θεός παρακολουθεί με τη σειρά του τη διαδικασία και μόλις αντιληφθεί ότι ο πράκτορας παίχτης έχει επιλέξει τους σωστούς συνδυασμούς και στα τρία δωμάτια, τότε ανοίγει την πόρτα προς το δωμάτιο 10 (κατάσταση Z).




**Μηχανισμός δωματίου 5**

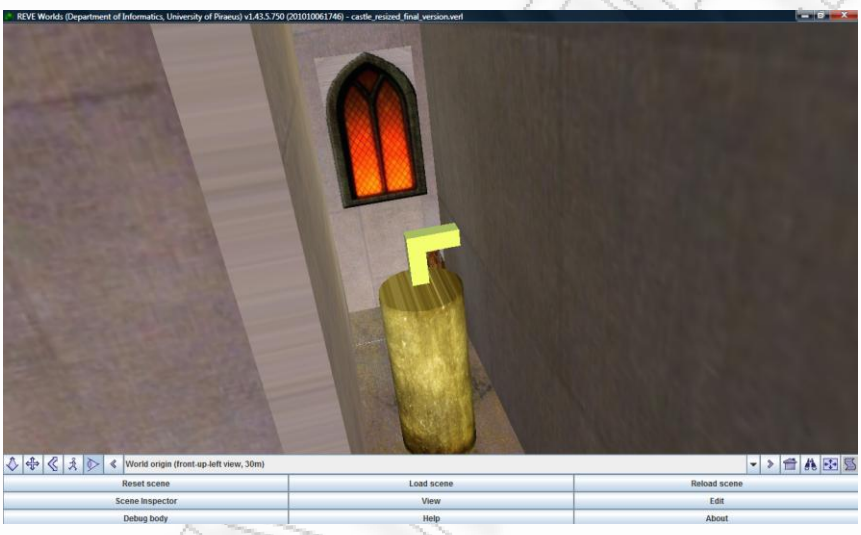



**Επιλογή σωστού κουμπιού στο συνδυασμό του δωματίου 6**

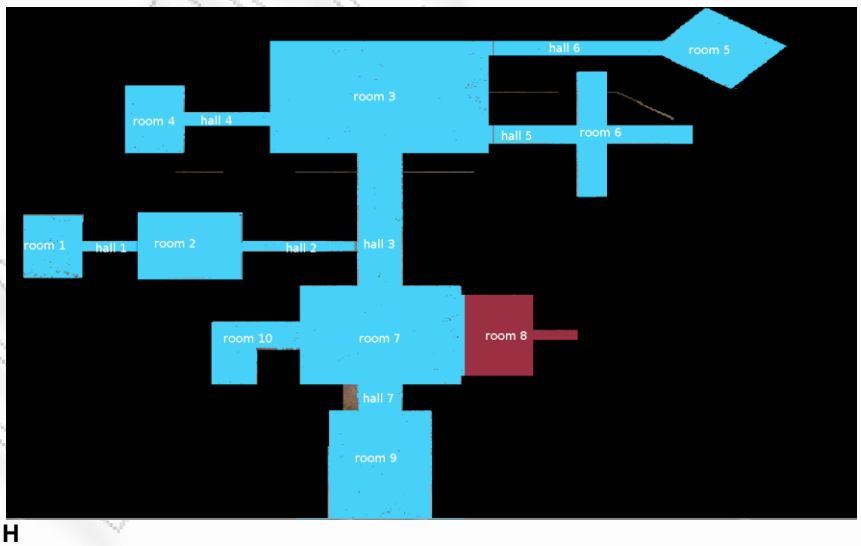



**Δωμάτιο7/9**

Στο δωμάτιο 9 υπάρχει ένα ακόμη μπαούλο που λειτουργεί όπως και αυτό του δωματίου 4. Ο πράκτορας παίχτης ανοίγει το μπαούλο, σπρώχνει τον πάτο που και με τη σειρά του ο πράκτορας θεός εμφανίζει έναν συνδυασμό γεωμετρικών σχημάτων/χρωμάτων αντίστοιχο με τον προηγούμενο, προσθέτοντας αυτή την φορά ένα ακόμη σύμβολο στον μηχανισμό. Στο δωμάτιο 10 υπάρχει έναν ακόμη γεωμετρικός μηχανισμός στον οποίο όμως ο πράκτορας παίχτης μπορεί να επιδράσει μόνον εάν έχει βρει τον χάρτη στο μπαούλο του δωματίου 9. Αυτό συμβαίνει γιατί στις προηγούμενες περιπτώσεις ο πράκτορας αποκτούσε πρόσβαση στα δωμάτια μόλις έβλεπε τον συνδυασμό. Αυτή την φορά η πόρτα δεν ανοίγει μόλις βρει τον καινούριο συνδυασμό αλλά μόλις λύσει τον προηγούμενο συνδυασμό. Αυτό μπορεί να έχει ως αποτέλεσμα ο πράκτορας να βρει τον μηχανισμό πριν από τον χάρτη να προσπαθήσει να συγκρίνει τα σύμβολα σε αυτόν αλλά χωρίς πληροφορία γεγονός που θα τον οδηγήσει σε exception. Μόλις λοιπόν ο πράκτορας παίχτης δει και τον χάρτη στο μπαούλο και τον μηχανισμό τότε μετακινεί το σύμβολο στην κορυφή του μηχανισμού, ο πράκτορας θεός αποκαλύπτει τα κουμπιά, ο πράκτορας παίχτης κάνει επιλογή με τον ίδιο τρόπο όπως και πριν και αν ο συνδυασμός είναι σωστός ο πράκτορας θεός θα ανοίξει την πόρτα για το δωμάτιο 8 (κατάσταση Η).



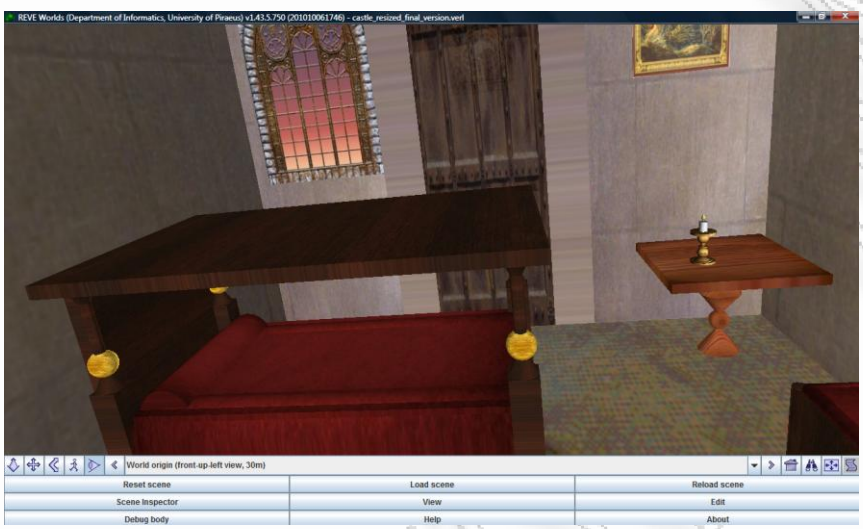

**Δωμάτιο 10**



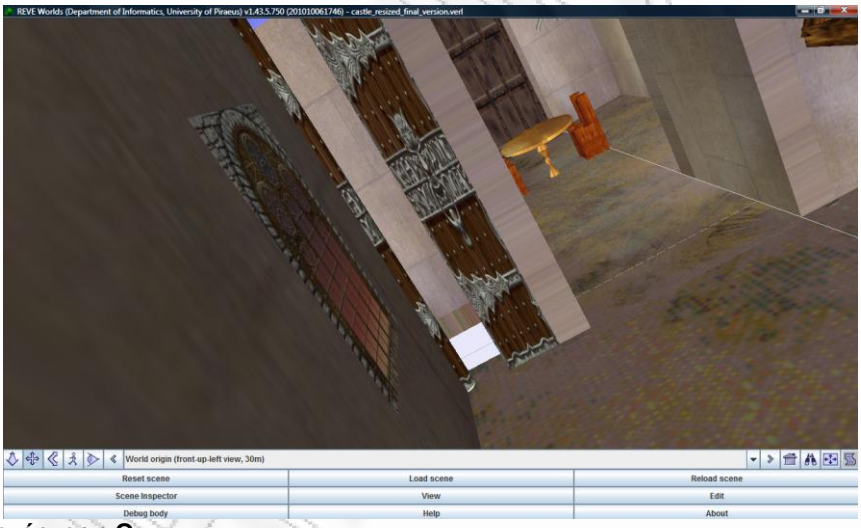

**Κατάσταση Η**

Στο δωμάτιο 8 βρίσκεται και ο τελευταίος συνδυασμός προκειμένου να απελευθερωθεί ο πράκτορας παίχτης από το κάστρο και να περάσει στην τελική κατάσταση Θ. Ο πράκτορας παίχτης πρέπει να εντοπίσει ένα μπαούλο το οποίο βρίσκεται μεν πίσω από μία πόρτα, η οποία όμως είναι κλάσης unreal οπότε για τους πράκτορες δεν υπάρχει οπότε δεν αποτελεί εμπόδιο. Ο πράκτορας παίχτης μόλις κάνει

sense το μπαούλο το ανοίγει, ο πράκτορας θεός εμφανίζει μέσα από το μπαούλο ένα σπαθί και ανοίγει την πόρτα της εξόδου.




**Δωμάτιο 8**




**Έξοδος – Κατάσταση Θ**

Μόλις ο πράκτορας παίχτης αντιληφθεί ότι η πόρτα της εξόδου είναι ανοιχτή σταματάει να κινείται, έχει πλέον απελευθερωθεί από το κάστρο και η εφαρμογή τερματίζεται.

## 7.2 Ο Πράκτορας HouseKeeper

Όπως έχουμε προαναφέρει ο πράκτορας housekeeper είναι ο πράκτορας θεός του κόσμου μας ο οποίος γνωρίζει τα πάντα για αυτόν και είναι υπεύθυνος για τις ενέργειες για τις οποίες δεν είναι άμεσα υπεύθυνος ο Sara agent. Είναι δηλαδή υπεύθυνος για τις ενέργειες για τις οποίες φαινομενικά γίνονται από μόνες τους, ως απόρροια μιας πράξης του Sara agent.

```
/** A housekeeper agent that is responsible for every interaction in the castle world. */
public class SampleHousekeeper1 extends BaseAgent {
……
……
}
```

Η λογική της λειτουργίας του housekeeper είναι η εξής: Ο πράκτορας συνδέεται στο port 4445. Κατά την διάρκεια της αρχικοποίησης του, δημιουργείται ένα frame στο οποίο ουσιαστικά θα καταγραφεί στη συνέχεια όλη η βάση γνώσης του housekeeper με μορφή η οποία μπορεί να γίνει κατανοητή από τον παρατηρητή, εμάς δηλαδή. Για την αρχικοποίηση του housekeeper και την δημιουργία του frame, του user interface ουσιαστικά και την σύνδεση στον AIP server χρησιμοποιείται ένας standard constructor, παρόμοιος με τους SampleHousekeeper1 (String name, String host, int port) και SampleHousekeeper2(String name, String host, int port), ο HouseKeeper_castle_version(String name, String host, int port).

```
/** The agent's GUI.*/
private SampleHousekeeper2Frame frame;

/**
* Standard constructor. Initializes the agent, creates the user-interface and connects to an AIP server.
* @param name the agent's name, as a {@link String}
* @param host the AIPd address to connect to, as a {@link String}
* @param port the APId port to connect to, as an int
*/
public HouseKeeper_castle_version(String name, String host, int port) {

        super(name);

        if (status.equals(Status.INITIALIZED)) {

                frame = new SampleHousekeeper2Frame(getName());
                frame.setSize(280, 320);
                frame.validate();
                frame.setLocationRelativeTo(null);
                frame.setVisible(true);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                frame.message("Connecting to " + host + ":" + port);
                try {
                        connect(host, port);
                } catch (IOException ex) {
                        frame.message("ERROR: Network error while connecting (" +
                                        ex.getMessage() + ")!");
                }
                if (status.equals(Status.CONNECTED)) {
                        frame.message("Connected.");
```

```
                }
        }
        frame.message("Close this window to terminate the application.");
}
```

Μετά την αρχικοποίηση του πράκτορα και την σύνδεση του σειρά έχει η δημιουργία της μεθόδου main(String[] args) η οποία δημιουργεί και χρησιμοποιεί τον πράκτορα της κλάσης HouseKeeper_castle_version χρησιμοποιώντας το public interface της κλάσης.

```
/** Main method, creates and uses an agent of the {@link HouseKeeper_castle_version} class using the
* class' public interface.
* @param args command-line arguments, not used by this implementation
*/
public static void main(String[] args) {

        // initialize the agent...
        HouseKeeper_castle_version agent = new HouseKeeper_castle_version("Housekeeper",
         "127.0.0.1", 4445);

        // check if the agent has been succesfully initialized...
        if (agent.getStatus().equals(Status.CONNECTED)) {

                // start the agent's behaviour...
                agent.start();
        } else {
                // initialization of the agent has failed, for some reason...
                System.out.println("ERROR: Agent initialization error!");
                return;
        }
}
```

Η μέθοδος main καλεί την getStatus(), οποία επιστρέφει μια enum type μεταβλητή η οποία εκφράζει την κατάσταση στην οποία βρίσκεται ο πράκτορας. Ένας housekeeper μπορεί να βρίσκεται σε μία από τις ακόλουθες καταστάσεις:

- CONNECTED
- CONNECTING
- EXECUTING
- HANDSHAKING
- INITIALIZED
- INITIALIZING
- SENSING
- STALE

Σε αυτό όμως το σημείο μας ενδιαφέρει εάν η κατάσταση του πράκτορα είναι CONNECTED και σε αυτή την περίπτωση καλείται η συνάρτηση start(). Αυτή η συνάρτηση, ελέγχει συνεχώς συγκεκριμένα items και τις συνθήκες στις οποίες βρίσκονται και όταν αυτές αλλάζουν αλλάζει και ο housekeeper την κατάσταση είτε του ίδιου item είτε κάποιου άλλου και καταγράφει στο user interface συνεχώς με την

συχνότητα που ο πράκτορας κάνει sense οι καταστάσεις στις οποίες είναι τα διάφορα items που μας ενδιαφέρουν. Η συνάρτηση start θα αναλυθεί παρακάτω. Σε περίπτωση όπου η κατάσταση του πράκτορα δεν είναι CONNECTED τότε εμφανίζεται μήνυμα λάθους και η κλάση τερματίζεται.

Για την ανάπτυξη του interface του frame εμφάνισης των μηνυμάτων δηλαδή χρησιμοποιήθηκαν οι παρακάτω κλάσεις οι οποίες όμως δεν θα αναλυθούν σε αυτό το σημείο καθώς δεν προσθέτουν κάποια πληροφορία για την λειτουργικότητα του housekeeper.

```
/** A GUI for agents of the {@link SampleHousekeeper2} class.*/
protected static class SampleHousekeeper2Frame extends JFrame {

private final JTextArea messages = new JTextArea();
private final JScrollPane jsp = new JScrollPane(messages);
private final DecimalFormat df = new DecimalFormat("0000");
private final long then = Calendar.getInstance().getTimeInMillis();

/** Initializes the GUI.*
* @param title the window title, as a {@link String}
*/
public SampleHousekeeper2Frame(String title) {
        setTitle(title);
        messages.setEditable(false);
        jsp.setBorder(new EmptyBorder(0, 0, 0, 0));
        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(jsp, BorderLayout.CENTER);
}

/** Adds a new message to the list of messages.*
* @param message the message to add, as a {@link String}
*/
public void message(String message) {
        long now = Calendar.getInstance().getTimeInMillis();
        String s = messages.getText();
        if (s != null && s.length() > 1024) {
                s = s.substring(s.length() - 1024, s.length());
                s = s.substring(s.indexOf("\n") + 1);
        }
        messages.setText(s + "[" + df.format((now - then) / 1000) + "] " + message + "\n");
        messages.setCaretPosition(messages.getDocument().getLength());
        }
}
```

Ένα πρόβλημα που έπρεπε να αντιμετωπιστεί κατά την υλοποίηση του housekeeper ήταν το πότε ακριβώς ο housekeeper θα πρέπει να επεμβαίνει στον κόσμο. Έστω ότι ο housekeeper κάνει συνεχώς sense σε ένα item και αυτό αλλάζει ξαφνικά κατάσταση λόγω της δράση του SARA agent. Παρατηρήθηκε ότι λόγω της αστραπιαίας αντίδρασης του housekeeper προκαλούνταν κάποια ενέργεια σε κάποιο item χωρίς όμως πρώτα να έχει ολοκληρώσει την δράση του ο SARA agent. Αυτό είχε ως αποτέλεσμα να μπερδεύει ουσιαστικά τον housekeeper ο οποίος ολοκλήρωνε την δράση του, έκανε ξανά

sense στο αρχικό item που δρούσε ο SARA agent, έβλεπε ότι δεν είναι στην σωστή θέση οπότε ξαναεκτελούσε την δράση του στο item με αποτέλεσμα να εκτελεί περισσότερες από μια φορές την δράση σε αυτό δημιουργώντας έτσι πρόβλημα λειτουργικότητας στον κόσμο. Σε αυτή την υλοποίηση ο housekeeper έκανε συνεχώς sense σε κάποιο αντικείμενο και έλεγχε την αλλαγή της π.χ. της θέσης του ελέγχοντας την απόλυτη μετακίνηση του. Για την αντιμετώπιση του προβλήματος ο housekeeper ελέγχει πλέον την σχετική μετακίνηση του αντικειμένου, ελέγχοντας τις αρχικές και τις τελικές συντεταγμένες του.

Ένα άλλο πρόβλημα κατά την αλλαγή θέσης των αντικειμένων το REVE Worlds έχει μια ελάχιστη απόκλιση από την τελική θέση η οποία όμως συνεχώς αυξανόμενη από τα διαδοχικά animations μπορεί να αυξηθεί και να δημιουργήσει πρόβλημα. Για αυτό το λόγω ορίστηκαν κάποια κατώφλια ελέγχου ώστε να μην επηρεάζονται οι πράξεις των πρακτόρων από αυτή τη διαφορά.

Επιπλέον πρέπει να τονίσουμε ότι οι συντεταγμένες του κάθε item διέπονται από το τοπικό σύστημα συνταγμένων του item και όχι από το σύστημα συντεταγμένων του κόσμου. Αυτό πρακτικά σημαίνει ότι οι άξονες ενός item που έχουν υποστεί περιστροφή δεν ταυτίζονται με τους άξονες του κόσμου. Αυτό δεν δημιουργεί πρόβλημα λειτουργικότητας στον κόσμο μας, αλλά μπορεί να δυσχεραίνει την προσπάθεια μας για την σωστή υλοποίηση της συμπεριφοράς του housekeeper, καθώς κάθε φορά θα πρέπει να ελέγχουμε το item στο οποίο θέλουμε δράσει ο πράκτορας.

Για να υλοποιηθεί όλη η λειτουργικότητα του κόσμου όπως αναλύθηκε παραπάνω στο κεφάλαιο του walkthrough δημιουργήθηκαν κάποιες συναρτήσεις για τον housekeeper για να αντιμετωπιστούν οι διάφορες περιπτώσεις δράσης του. Πιο συγκεκριμένα χρειάστηκε να δημιουργηθούν συναρτήσεις για τις ακόλουθες ενέργειες:

- Συναρτήσεις για items τα οποία πραγματοποιούν κατά την κίνηση τους translation
- Συναρτήσεις για items τα οποία πραγματοποιούν κατά την κίνηση τους rotation
- Συναρτήσεις για κόμβους items τα οποία πραγματοποιούν κατά την κίνηση τους translation
- Συναρτήσεις για κόμβους items τα οποία πραγματοποιούν κατά την κίνηση τους rotation

Αυτές οι συναρτήσεις θα κληθούν στη συνέχεια από την συνάρτηση start() η οποία υλοποιεί ουσιαστικά την συμπεριφορά του housekeeper. Η start() δηλαδή δεν είναι τίποτα άλλο παρά μια συλλογή των παραπάνω συναρτήσεων για κάθε ένα item όπου μπορεί να επιδράσει ο housekeeper ή που παρακολουθεί.

Οι παραπάνω συναρτήσεις θα παρουσιαστούν αναλυτικά αμέσως παρακάτω, πριν όμως αναλυθούν θα παρουσιαστούν κάποιες συναρτήσεις οι οποίες χρησιμοποιήθηκαν για την υλοποίηση τους και οι οποίες ανήκουν στην βιβλιοθήκη του Housekeeper.

Η πρώτη από αυτές είναι η getKB(), η οποία επιστρέφει την γνώση ουσιαστικά που έχει εκείνη τη στιγμή ο housekeeper στη βάση γνώσης του. Για να λάβουμε τώρα πληροφορία για ένα συγκεκριμένο item ή symbol στη βάση γνώσης χρησιμοποιούμε την get(String symbol, String...args). Τέλος προκειμένου να εκτελέσει ο housekeeper μια ενέργεια χρησιμοποιούμε την συνάρτηση act() η οποία είναι Boolean και επιστρέφει true ή false ανάλογα με την επιτυχή ή μη εκτέλεση της εντολής.

### 7.2.1 Συναρτήσεις για items τα οποία πραγματοποιούν κατά την κίνηση τους translation

Μια βασική λειτουργία την οποία καλείται να εκτελέσει ο housekeeper είναι η μετατόπιση ολόκληρου item ως αποτέλεσμα μιας δράσης του SARA agent. Χαρακτηριστικό παράδειγμα αυτής της λειτουργικότητας είναι η αλλαγή θέσης ενός διακόπτη η οποία προκαλεί το άνοιγμα μιας πόρτας από τον housekeeper.

Αρχικά ορίζουμε ένα ελάχιστο κατώφλι για τον έλεγχο της μετατόπισης και τρεις καταστάσεις για το item, εάν είναι «κλειστό», «ανοιχτό» ή αν η κατάσταση του είναι απροσδιόριστη. Ένα item χαρακτηρίζεται ως «κλειστό» εάν οι συντεταγμένες του είναι οι ίδιες ή διαφέρουν κατά το ελάχιστο κατώφλι με τις αρχικές συντεταγμένες κατά την αρχικοποίηση του κόσμου. Ένα item χαρακτηρίζεται ως «ανοιχτό» εάν οι συντεταγμένες του είναι ίδιες ή διαφέρουν κατά το ελάχιστο κατώφλι με τις συντεταγμένες τις οποίες ένας πράκτορας μετατόπισε το item. Τέλος ένα item χαρακτηρίζεται ως «απροσδιόριστο» σε περίπτωση όπου οι συντεταγμένες του δεν ταυτίζονται με καμία από τις παραπάνω περιπτώσεις.

```
public static float MIN_FLOAT = 0.1001f;
public static int STATE_OPEN = 1;
public static int STATE_CLOSED = -1;
public static int STATE_UNDEFINED = 0;
```

Επόμενο βήμα είναι ο υπολογισμός της θέσης του item σε σχέση και με το ελάχιστο κατώφλι που ορίσαμε. Για αυτό ορίζουμε την συνάρτηση isEqual () η οποία είναι τύπου Boolean και η οποία δέχεται ως ορίσματα δύο διαφορετικές συντεταγμένες του ίδιου item τις αρχικές και τις τελικές του και επιστρέφει true εάν η απόλυτη τιμή της διαφοράς των αρχικών και τελικών συντεταγμένων είναι μικρότερη από το ελάχιστο κατώφλι που ορίσαμε.

```
public boolean isEqual(float x0, float y0, float z0, float x1, float y1, float z1) {
        return Math.abs(x1 - x0) < MIN_FLOAT &&
        Math.abs(y1 - y0) < MIN_FLOAT &&
        Math.abs(z1 - z0) < MIN_FLOAT;
}
```

Επόμενο βήμα είναι ο προσδιορισμός της κατάστασης του item. Η συνάρτηση state() δέχεται ως ορίσματα το item για το οποίο ενδιαφερόμαστε και τις αρχικές και τελικές συντεταγμένες του όπως αυτές θα πρέπει να είναι. Δηλαδή οι αρχικές συντεταγμένες είναι οι συντεταγμένες του item κατά την αρχικοποίηση του κόσμου και οι τελικές συντεταγμένες είναι οι συντεταγμένες που θα πρέπει να έχει το item μετά από μια συγκεκριμένη ενέργεια ενός πράκτορα. Αρχικά εξάγουμε τις πραγματικές συντεταγμένες του item και στη συνέχεια με την βοήθεια της συνάρτησης isEqual ελέγχουμε εάν το item είναι open, closed ή undefined.

```
public int state(String itemName, float xo, float yo, float zo, float xc, float yc, float zc) {

        // get the location symbol instances for the specified item...
        Vector<Vector<String>> locations = getKB().get("location", itemName);

        // get the second argument of the first symbol instance found...
        String location = locations.get(0).get(1);

        // the argument is of the form "x,y,z", parse it to get...
        // individual coordinates...
        String[] coords = location.split(",");
        float x = Float.valueOf(coords[0]);
        float y = Float.valueOf(coords[1]);
        float z = Float.valueOf(coords[2]);

        // compare...
        if (isEqual(x, y, z, xo, yo, zo)) {
                return STATE_OPEN;
        }
        else if (isEqual(x, y, z, xc, yc, zc)) {
                return STATE_CLOSED;
```

```
        }
        else {
                return STATE_UNDEFINED;
        }
}
```

Στη συνέχεια για την ολοκλήρωση της ενέργειας είναι απαραίτητος ο ορισμός της συνάρτησης move ώστε ο housekeeper να εκτελέσει την ενέργεια της μετατόπισης πάνω στο item. Ορίσματα της συνάρτησης move() είναι το όνομα του item, το όνομα του accesspoint στο οποίο θα κάνει attach ο housekeeper, το όνομα της συνάρτησης την οποία θα εκτελέσει ο housekeeper και τις σχετικές συντεταγμένες μετατόπισης του item. Ο housekeeper κάνει attach τον hand effector του στο accespoint του item, στη συνέχεια επιλέγει την συνάρτηση που ορίσαμε για να εκτελέσει την μετατόπιση, εκτελεί την σχετική μετατόπιση και κάνει detach. Η συνάρτηση move() είναι boolean.

```
public boolean move(String itemName, String accessPointName, String functionName, float x, float y, float z) {
        if (!act("attach hand " + itemName + " " + accessPointName)) {
                frame.message("ERROR: Could not attach hand effector to " + accessPointName + "
                accesspoint!");
                return false;
        }
        if (!act("select hand " +  " move " + functionName)) {
                frame.message("ERROR: Could not select " + functionName + " function for move
                action!");
                return false;
        }
        if (!act("act hand "  + " move " + x + "," + y + "," + z + " true")) {
                frame.message("ERROR: Could not move " + itemName + "!");
                return false;
        }
        if (!act("attach hand " )) {
                frame.message("ERROR: Could not dettach hand effector!");
                return false;
        }
return true;
}
```

Τελευταίο βήμα αποτελεί ο ορισμός των συναρτήσεων ensureOpen και ensureClosed όπου υπολογίζονται οι συντεταγμένες του item και συγκρίνονται με τις αρχικές και τελικές συντεταγμένες του item. Ορίσματα και των δύο συναρτήσεων είναι το όνομα του item, το accesspoint του και το όνομα της function του και τέλος οι αρχικές και οι τελικές του συντεταγμένες αντίστοιχα, και η σχετική μετατόπιση του item. Στην ensureClosed οι συντεταγμένες του item συγκρίνονται με τις τελικές συντεταγμένες του με την βοήθεια της isEqual() και σε περίπτωση όπου είναι ίσες τότε καλείται η move() για να μετατοπίσει το item στις αρχικές του συντεταγμένες. Αντίθετα στην ensureClosed οι συντεταγμένες του item συγκρίνονται με τις αρχικές συντεταγμένες του με την βοήθεια της equal() και σε περίπτωση όπου είναι ίσες τότε καλείται η move() για να μετατοπίσει το item στις τελικές του συντεταγμένες.

```
public void ensureOpen(String itemName, String accessPointName, String functionName, float xo, float
```

```
yo, float zo, float xc, float yc, float zc) {

        // get the location symbol instances for the specified item...
        Vector<Vector<String>> locations = getKB().get("location", itemName);

        // get the second argument of the first symbol instance found...
        String location = locations.get(0).get(1);

        // the argument is of the form "x,y,z", parse it to get...
        // individual coordinates...
        String[] coords = location.split(",");
        float x = Float.valueOf(coords[0]);
        float y = Float.valueOf(coords[1]);
        float z = Float.valueOf(coords[2]);

        // compare...
        if (isEqual(x, y, z, xo, yo, zo)) {
                // the specified item is already in the "open" state, do nothing...
        }
        else if (isEqual(x, y, z, xc, yc, zc)) {
                move(itemName, accessPointName, functionName, xo, yo, zo);
        }
        else {
                // the specified item is in an undefined state, do nothing...
        }
}
```

```
public void ensureClosed(String itemName, String accessPointName, String functionName, float xo, float
yo, float zo, float xc, float yc, float zc) {

        // get the location symbol instances for the specified item...
        Vector<Vector<String>> locations = getKB().get("location", itemName);

        // get the second argument of the first symbol instance found...
        String location = locations.get(0).get(1);

        // the argument is of the form "x,y,z", parse it to get...
        // individual coordinates...
        String[] coords = location.split(",");
        float x = Float.valueOf(coords[0]);
        float y = Float.valueOf(coords[1]);
        float z = Float.valueOf(coords[2]);

        // compare...
        if (isEqual(x, y, z, xo, yo, zo)) {
```

```
                move(itemName, accessPointName, functionName, xc, yc, zc);
        }
        else if (isEqual(x, y, z, xc, yc, zc)) {
                // the specified item is already in the "closed" state, do nothing...
        }
        else {
                // the specified item is in an undefined state, do nothing...
        }
}
```

Παράδειγμα εφαρμογής των παραπάνω αποτελεί το παρακάτω παράδειγμα όπου ο housekeeper ελέγχει εάν υπάρχουν στον κόσμο τα items Door1 και Door1opener. Σε περίπτωση όπου ο SARA agent ανοίξει τον διακόπτη Door1opener ο housekeeper φροντίζει να ανοίξει την πόρτα Door1 και αντίθετα εάν ο SARA agent κλείσει τον διακόπτη Door1opener ο housekeeper φροντίζει να κλείσει την πόρτα Door1. Εάν ο διακόπτης βρίσκεται σε απροσδιόριστη θέση τότε ο housekeeper δεν εκτελεί καμία ενέργεια.

```
if (getKB().get("item", "Door1").size() != 0 && getKB().get("item", "Door1opener").size() != 0) {

        if (state("Door1opener", 0.4f,1, 10, 0.6f, 1, 10) == STATE_CLOSED) {
                frame.message("Door1opener: Run State_CLOSED");
                ensureClosed("Door1", "moverDoor", "moveDoor", -4.9f, 0, 3, 2.5f,0,0);
                frame.message("Door1: Run State_CLOSED");
        }

        else if (state("Door1opener", 0.4f,1, 10, 0.6f, 1, 10) == STATE_OPEN) {
                frame.message("Door1opener: Run State_OPEN!");
                ensureOpen("Door1", "moverDoor", "moveDoor",-2.5f,0,0, -4.9f, 0, 5.5f);
                frame.message("Door1: Run State_OPEN!");
        }
        else if (state("Door1opener", 0.4f,1, 10, 0.6f, 1, 10) == STATE_UNDEFINED) {
                frame.message("Door1opener: Run STATE_UNDEFINED!");
        }
        else {
        }
}
```

### 7.2.2 Συναρτήσεις για items τα οποία πραγματοποιούν κατά την κίνηση τους rotation

Επόμενη βασική λειτουργία την οποία καλείται να εκτελέσει ο housekeeper είναι η περιστροφή ολόκληρου item ως αποτέλεσμα μιας δράσης του SARA agent. Χαρακτηριστικό παράδειγμα αυτής της λειτουργικότητας είναι η αλλαγή θέσης ενός διακόπτη η οποία προκαλεί την περιστροφή μιας πόρτας από τον housekeeper ώστε να την ανοίξει.

Αρχικά ορίζουμε ένα ελάχιστο κατώφλι για τον έλεγχο της περιστροφής και τρεις καταστάσεις για το item, εάν είναι «κλειστό», «ανοιχτό» ή αν η κατάσταση του είναι απροσδιόριστη ακριβώς όπως και στην μετατόπιση ενός item από τον housekeeper.

```
public static float MIN_FLOAT_ROTATE = 0.2f;
public static int STATE_OPEN_ROTATED = 1;
public static int STATE_CLOSED_ROTATED = -1;
public static int STATE_UNDEFINED_ROTATED = 0;
```

Επόμενο βήμα είναι ο υπολογισμός της θέσης του item σε σχέση και με το ελάχιστο κατώφλι που ορίσαμε. Για αυτό ορίζουμε την συνάρτηση isEqual_ROTATED () κατά την isEqual η οποία είναι τύπου Boolean και η οποία δέχεται ως ορίσματα δύο διαφορετικές συντεταγμένες του ίδιου item τις αρχικές και τις τελικές του και επιστρέφει true εάν η απόλυτη τιμή της εφαπτομένης τόξου της γωνίας που δημιουργείται ανάμεσα στις αρχικές και τελικές συντεταγμένες στον άξονα περιστροφής του item είναι μεγαλύτερη από το ελάχιστο κατώφλι που ορίσαμε.

```
public boolean isEqual_ROTATED(float x0, float y0, float z0, float x1, float y1, float z1) {
        return Math.abs((Math.atan2(y1 - y0, x1 - x0))) > MIN_FLOAT_ROTATE ||
        Math.abs((Math.atan2(y1 - y0, z1 - z0))) > MIN_FLOAT_ROTATE ||
        Math.abs((Math.atan2(x1 - x0, z1 - z0))) > MIN_FLOAT_ROTATE;
}
```

Επόμενο βήμα είναι ο προσδιορισμός της κατάστασης του item. Για αυτό ορίζουμε την συνάρτηση state_ROTATED() κατά την state() η οποία δέχεται ως ορίσματα το item για το οποίο ενδιαφερόμαστε και τις αρχικές και τελικές συντεταγμένες του όπως αυτές θα πρέπει να είναι. Έτσι υπολογίζουμε τις πραγματικές συντεταγμένες του item και στη συνέχεια με την βοήθεια της συνάρτησης isEqual_ROTATED ελέγχουμε εάν το item είναι open, closed ή undefined.

```
public int state_ROTATED(String itemName, float xo, float yo, float zo, float xc, float yc, float zc) {

        // get the location symbol instances for the specified item...
        Vector<Vector<String>> locations = getKB().get("location", itemName);

        // get the second argument of the first symbol instance found...
        String location = locations.get(0).get(1);

        // the argument is of the form "x,y,z", parse it to get...
        // individual coordinates...
        String[] coords = location.split(",");
        float x = Float.valueOf(coords[0]);
        float y = Float.valueOf(coords[1]);
        float z = Float.valueOf(coords[2]);

        // compare...
        if (isEqual_ROTATED(x, y, z, xo, yo, zo)) {
                return STATE_OPEN_ROTATED;
        }
        else if (isEqual_ROTATED(x, y, z, xc, yc, zc)) {
                return STATE_CLOSED_ROTATED;
        }
```

```
        else {
                return STATE_UNDEFINED_ROTATED;
        }
}
```

Στη συνέχεια για την ολοκλήρωση της ενέργειας είναι απαραίτητος ο ορισμός της συνάρτησης turn ώστε ο housekeeper να εκτελέσει την ενέργεια της περιστροφής πάνω στο item. Ορίσματα της συνάρτησης turn() είναι το όνομα του item, το όνομα του accesspoint στο οποίο θα κάνει attach ο housekeeper, το όνομα της συνάρτησης την οποία θα εκτελέσει ο housekeeper, τους άξονες περιστροφής του item και την γωνία περιστροφής σε ακτίνια. Ο housekeeper κάνει attach τον hand effector του στο accespoint του item, στη συνέχεια επιλέγει την συνάρτηση που ορίσαμε για να εκτελέσει την περιστροφή, εκτελεί την περιστροφή και κάνει detach. Η συνάρτηση turn() είναι boolean.

```
public boolean turn (String itemName, String accessPointName, String functionName, float xa, float ya,
float za, float angle) {
        if (!act("attach hand " + itemName + " " + accessPointName)) {
                frame.message("ERROR: Could not attach hand effector to " + accessPointName + "
                accesspoint!");
                return false;
        }
        if (!act("select hand " +  " turn " + functionName)) {
                frame.message("ERROR: Could not select " + functionName + " function for turn
                action!");
                return false;
        }
        if (!act("act hand "  + " turn " + xa + "," + ya + "," + za + ","+ angle)) {
                frame.message("ERROR: Could not turn " + itemName + "!");
                return false;
        }
        if (!act("attach hand " )) {
                frame.message("ERROR: Could not dettach hand effector!");
                return false;
        }
        return true
}
```

Τελευταίο βήμα αποτελεί ο ορισμός των συναρτήσεων ensureOpen_ROTATED και ensureClosed_ROTATED όπου υπολογίζονται οι συντεταγμένες του item και συγκρίνονται με τις αρχικές και τελικές συντεταγμένες. Ορίσματα και των δύο συναρτήσεων είναι το όνομα του item, το accesspoint του και το όνομα της function του και τέλος οι αρχικές και οι τελικές του συντεταγμένες, οι άξονες περιστροφής και η γωνία περιστροφής σε ακτίνια. Στην ensureOpen_ROTATED οι συντεταγμένες του item συγκρίνονται με τις τελικές συντεταγμένες του με την βοήθεια της isequal_ROTATED() και σε περίπτωση όπου είναι ίσες τότε καλείται η turn() για να περιστρέψει το item στις αρχικές του συντεταγμένες. Αντίθετα στην ensureClosed_ROTATED οι συντεταγμένες του item συγκρίνονται με τις αρχικές συντεταγμένες του με την βοήθεια της isequal_ROTATED() και σε περίπτωση όπου είναι ίσες τότε καλείται η turn() για να περιστρέψει το item στις τελικές του συντεταγμένες.

```
public void ensureOpen_ROTATED(String itemName, String accessPointName, String functionName,
float xo, float yo, float zo, float xc, float yc, float zc, float xa, float ya, float za,float angle) {

        // get the location symbol instances for the specified item...
        Vector<Vector<String>> locations = getKB().get("location", itemName);

        // get the second argument of the first symbol instance found...
        String location = locations.get(0).get(1);

        // the argument is of the form "x,y,z", parse it to get...
        // individual coordinates...
        String[] coords = location.split(",");
        float x = Float.valueOf(coords[0]);
        float y = Float.valueOf(coords[1]);
        float z = Float.valueOf(coords[2]);

        // compare...

        double angle0= (Math.atan2(y - yc, x - xc));
        frame.message("Angle:  " + angle0 + "!");

        if (isEqual(x, y, z, xo, yo, zo)) {
                // the specified item is already in the "open" state, do nothing...
        }
        else if (isEqual(x, y, z, xc, yc, zc)) {
                turn (itemName, accessPointName, functionName, xa, ya, za, angle);
        }
        else {
                // the specified item is in an undefined state, do nothing...
        }
}
```

```
public void ensureClosed_ROTATED(String itemName, String accessPointName, String functionName,
float xo, float yo, float zo,float xc, float yc, float zc, float xa, float ya, float za, float angle) {

        // get the location symbol instances for the specified item...
        Vector<Vector<String>> locations = getKB().get("location", itemName);

        // get the second argument of the first symbol instance found...
        String location = locations.get(0).get(1);

        // the argument is of the form "x,y,z", parse it to get...
        // individual coordinates...
        String[] coords = location.split(",");
        float x = Float.valueOf(coords[0]);
```

```
        float y = Float.valueOf(coords[1]);
        float z = Float.valueOf(coords[2]);

        // compare...

        double angle0= (Math.atan2(y - yo, x - xo));
        frame.message("Angle:  " + angle0 + "!");

        if (isEqual(x, y, z, xo, yo, zo)) {
                turn (itemName, accessPointName, functionName, xa, ya, za, angle);
        }
        else if (isEqual(x, y, z, xc, yc, zc)) {
                // the specified item is already in the "closed" state, do nothing...
        }
        else {
                // the specified item is in an undefined state, do nothing...
        }
}
```

Παράδειγμα εφαρμογής των παραπάνω αποτελεί το παρακάτω παράδειγμα όπου ο housekeeper ελέγχει εάν υπάρχουν στον κόσμο τα items sword και exit_door. Σε περίπτωση όπου ο SARA agent ανοίξει τον διακόπτη sword ο housekeeper φροντίζει να περιστρέψει την πόρτα exit_Door ώστε να την ανοίξει.

```
if ((getKB().get("item", "sword").size()!=0)&&(getKB().get("item", "exit_door").size()!=0)) {
        if (state("sword", 35.3f, 1.0999999f, 19.1f, 35.3f, 0.6f, 19.1f) == STATE_CLOSED) {
                frame.message("sword==STATE_CLOSED");
        }else if(state("sword",  35.3f, 1.0999999f, 19.1f, 35.3f, 0.6f, 19.1f) == STATE_OPEN){
                frame.message("sword==STATE_OPEN");
                ensureOpen_ROTATED("exit_door", "exit_door_access_p", "rotate",19.939806f, 0.0f,
                25.695646f, 19.8f, 0, 25.6f, 0,1,0,-1.2f);
        }else if (state("sword",  35.3f, 1.0999999f, 19.1f, 35.3f, 0.6f, 19.1f) == STATE_UNDEFINED){
                frame.message("sword==STATE_UNDEFINED");
        }else{
        }
}
```

### 7.2.3 Συναρτήσεις για κόμβους items τα οποία πραγματοποιούν κατά την κίνηση τους translation

Πέρα από τις ενέργειες τις οποίες καλείται να εκτελέσει ο housekeeper σε ολόκληρα items του κόσμου, καλείται να δράσει και σε κόμβους των items προσπαθώντας να τους μετατοπίσει. Χαρακτηριστικό παράδειγμα αυτής της λειτουργικότητας είναι η εμφάνιση του χάρτη στα μπαούλα όταν ο SARA agent τα ανοίξει.

Και σε αυτή την περίπτωση πρέπει να υπάρχει ένα ελάχιστο κατώφλι για τον έλεγχο της μετατόπισης του κόμβου και τρεις καταστάσεις για το εάν είναι «κλειστός», «ανοιχτός» ή αν η κατάσταση του είναι απροσδιόριστη. Για αυτόν τον λόγο επαναχρησιμοποιούνται οι μεταβλητές που χρησιμοποιήσαμε για τις συναρτήσεις για items που πραγματοποιούν μετατόπιση

```
public static float MIN_FLOAT = 0.1001f;
public static int STATE_OPEN = 1;
public static int STATE_CLOSED = -1;
public static int STATE_UNDEFINED = 0;
```

Με το ίδιο σκεπτικό επαναχρησιμοποιείται και η συνάρτηση isEqual() όπως ακριβώς αυτή ορίστηκε παραπάνω στις συναρτήσεις για items που πραγματοποιούν μετατόπιση για να βρούμε την θέση του κόμβου σε σχέση και με το ελάχιστο κατώφλι που ορίσαμε.

```
public boolean isEqual(float x0, float y0, float z0, float x1, float y1, float z1) {
        return Math.abs(x1 - x0) < MIN_FLOAT &&
        Math.abs(y1 - y0) < MIN_FLOAT &&
        Math.abs(z1 - z0) < MIN_FLOAT;
}
```

Επόμενο βήμα είναι ο προσδιορισμός της κατάστασης του κόμβου του item. Για αυτό ορίζουμε την συνάρτηση state_nodes() η οποία δέχεται ως ορίσματα ένα σύμβολο που έχουμε ορίσει στον κόμβο για τον οποίο ενδιαφερόμαστε και τις αρχικές και τελικές συντεταγμένες του όπως αυτές θα πρέπει να είναι. Όπως έχουμε προαναφέρει ορίζοντας σύμβολα στο semantic model μία από τις πληροφορίες οι οποίες γίνονται διαθέσιμες προς αντίληψη στον πράκτορα είναι η θέση του item/ κόμβου. Αυτή την λειτουργικότητα εκμεταλλευόμαστε εδώ ώστε να μπορέσουμε να προσδιορίσουμε την κατάσταση στην οποία βρίσκεται ο κόμβος.

```
public int state_nodes(String symbolName, float xo, float yo, float zo, float xc, float yc, float zc) {

        // get the location symbol instances for the specified node...
        Vector<Vector<String>> locations = getKB().get(symbolName);

        // get the second argument of the first symbol instance found...
        String location = locations.get(0).get(1);
        // the argument is of the form "x,y,z", parse it to get...
        // individual coordinates...
        String[] coords = location.split(",");
        float x = Float.valueOf(coords[0]);
        float y = Float.valueOf(coords[1]);
        float z = Float.valueOf(coords[2]);

        // compare...
        if (isEqual(x, y, z, xo, yo, zo)) {
                return STATE_OPEN;
        }
        else if (isEqual(x, y, z, xc, yc, zc)) {
                return STATE_CLOSED;
        }
        else {
```

```
                return STATE_UNDEFINED;
        }
}
```

Στη συνέχεια για την ολοκλήρωση της ενέργειας είναι απαραίτητος ο ορισμός της συνάρτησης move_nodes ώστε ο housekeeper να εκτελέσει την ενέργεια της μετατόπισης πάνω στον κόμβο του item. Ορίσματα της συνάρτησης move_nodes() είναι το όνομα του item, το όνομα του accesspoint στο οποίο θα κάνει attach ο housekeeper, το όνομα της συνάρτησης την οποία θα εκτελέσει ο housekeeper και τις σχετικές συντεταγμένες μετατόπισης του item. Ο housekeeper κάνει attach τον hand effector του στο accespoint του item, στη συνέχεια επιλέγει την συνάρτηση που ορίσαμε για να εκτελέσει την μετατόπιση, εκτελεί την σχετική μετατόπιση και κάνει detach. Η συνάρτηση move() είναι boolean.

```
public boolean move_nodes(String itemName, String accessPointName, String functionName, float
x,float y, float z) {
        if (!act("attach hand " + itemName + " " + accessPointName)) {
                frame.message("ERROR: Could not attach hand effector to " + accessPointName + "
                accesspoint!");
        return false;
        }
        if (!act("select hand " +  " use " + functionName)) {
                frame.message("ERROR: Could not select " + functionName + " function for move
                action!");
                return false;
        }
        if (!act("act hand "  + " use " + x + "," + y + "," + z )) {
                frame.message("ERROR: Could not move " + itemName + "!");
                return false;
        }
        if (!act("attach hand " )) {
                frame.message("ERROR: Could not dettach hand effector!");
                return false;
        }
        return true;
}
```

Τελευταίο βήμα αποτελεί ο ορισμός των συναρτήσεων ensureOpen_nodes και ensureClosed_nodes όπου υπολογίζονται οι συντεταγμένες του item και συγκρίνονται με τις αρχικές και τελικές συντεταγμένες του item. Ορίσματα και των δύο συναρτήσεων είναι το όνομα του item, το σύμβολο που έχουμε ορίσει στον κόμβο, το accesspoint του και το όνομα της function του και τέλος οι αρχικές ή οι τελικές του συντεταγμένες αντίστοιχα, και η σχετική μετατόπιση του item. Στην ensureClosed_nodes οι συντεταγμένες του item συγκρίνονται με τις τελικές συντεταγμένες του με την βοήθεια της isEqual() και σε περίπτωση όπου είναι ίσες τότε καλείται η move_nodes() για να μετατοπίσει τον κόμβο στις αρχικές του συντεταγμένες. Αντίθετα στην ensureClosed_nodes οι συντεταγμένες του κόμβου του item συγκρίνονται με τις αρχικές συντεταγμένες του με την βοήθεια της isEqual() και σε περίπτωση όπου είναι ίσες τότε καλείται η move_nodes() για να μετατοπίσει τον κόμβο του item στις τελικές του συντεταγμένες.

```
public void ensureOpen_nodes(String itemName, String symbolName,String accessPointName, String functionName, float xo, float yo, float zo, float xc, float yc, float zc) {

	// get the location symbolName instances for the specified item...
	Vector<Vector<String>> locations = getKB().get(symbolName);

	// get the second argument of the first symbolName instance found...
	String location = locations.get(0).get(1);

	// the argument is of the form "x,y,z", parse it to get...
	// individual coordinates...
	String[] coords = location.split(",");
	float x = Float.valueOf(coords[0]);
	float y = Float.valueOf(coords[1]);
	float z = Float.valueOf(coords[2]);

	// compare...
	if (isEqual(x, y, z, xo, yo, zo)) {
		// the specified item is already in the "open" state, do nothing...
	}
	else if (isEqual(x, y, z, xc, yc, zc)) {
		move_nodes(itemName, accessPointName, functionName, xo, yo, zo);
	}
	else {
		// the specified item is in an undefined state, do nothing...
	}
}
```

```
public void ensureClosed_nodes(String itemName, String symbolName, String accessPointName, String functionName, float xo, float yo, float zo, float xc, float yc, float zc) {

	// get the location symbolName instances for the specified item...
	Vector<Vector<String>> locations = getKB().get(symbolName);

	// get the second argument of the first symbolName instance found...
	String location = locations.get(0).get(1);

	// the argument is of the form "x,y,z", parse it to get...
	// individual coordinates...
	String[] coords = location.split(",");
	float x = Float.valueOf(coords[0]);
	float y = Float.valueOf(coords[1]);
	float z = Float.valueOf(coords[2]);

	// compare...
```

```
        if (isEqual(x, y, z, xo, yo, zo)) {
                move_nodes(itemName, accessPointName, functionName, xc, yc, zc);
        }
        else if (isEqual(x, y, z, xc, yc, zc)) {
                // the specified item is already in the "closed" state, do nothing...
        }
        else {
                // the specified item is in an undefined state, do nothing...
        }
}
```

Παράδειγμα εφαρμογής των παραπάνω αποτελεί το παρακάτω παράδειγμα όπου ο housekeeper ελέγχει εάν υπάρχουν στον κόσμο το item chest_room4. Σε περίπτωση όπου ο SARA agent ανοίξει το μπαούλο μετακινώντας τον κόμβο _in_up ο housekeeper φροντίζει να εμφανίσει τον χάρτη μετατοπίζοντας τον κόμβο Hidden_map. Επιπλέον μόλις ολοκληρώσει την μετατόπιση του χάρτη φροντίζει να ανοίξει τις πόρτες door6,door7 και door3 για τα κλειδωμένα δωμάτια.

```
if (getKB().get("item", "chest_room4").size() != 0 ) {
        if (state_nodes("chestr4_in_up_symbol",  0,1, 0 , 0, 2.5f,0) == STATE_CLOSED){
                frame.message("chestr4_in_up_symbol: Run State_CLOSED");
        }
        else if (state_nodes("chestr4_in_up_symbol",  0,1, 0 , 0, 2.5f,0) == STATE_OPEN){
                frame.message("chestr4_in_up_symbol: Run State_OPEN");
                ensureOpen_nodes("chest_room4","chestr4_Hidden_map_symbol", "chest_r4_map_p",
                 "translate", 0,-2,0, 0, -3, 0);
        }else if (state_nodes("chestr4_in_up_symbol",  0,1, 0 , 0, 2.5f,0) == STATE_UNDEFINED){
                frame.message("chestr4_in_up_symbol: Run State_UNDEFINED");
        }else{
                //do nothing
        }
        if (state_nodes( "chestr4_Hidden_map_symbol",  0,-2,0, 0, -3, 0) == STATE_CLOSED){
                frame.message("chestr4_Hidden_map_symbol: Run State_CLOSED");
        }
        else if (state_nodes("chestr4_Hidden_map_symbol",  0,-2,0, 0, -3, 0) == STATE_OPEN){
                frame.message("chestr4_Hidden_map_symbol: Run State_CLOSED");
                ensureOpen("door6", "Door6_access_p", "translate", -1.5f,0,0, 30.5f, 0, -5);
                ensureOpen("door7", "Door7_access_p", "translate", -2,0,0, 30.5f, 0, -10);
                ensureOpen("door3", "Door3_access_p", "translate", -2,0,0,22.1f, 0, 14.5f);
                }else if (state_nodes("chestr4_Hidden_map_symbol",  0,-2,0, 0, -3, 0)
                 ==STATE_UNDEFINED){
                        frame.message("chestr4_Hidden_map_symbol: Run State_UNDEFINED");
                }else{
                        //do nothing
        }
}
```

### 7.2.4 Συναρτήσεις για κόμβους items τα οποία πραγματοποιούν κατά την κίνηση τους rotation

Τελευταίο παράδειγμα λειτουργικότητας το οποίο καλείται να εκτελέσει ο housekeeper είναι η περιστροφή κόμβων των items. Χαρακτηριστικό παράδειγμα αυτής της λειτουργικότητας αποτελεί το άνοιγμα του συρταριού στο δωμάτιο 3 ώστε να εμφανιστεί το κλειδί που ανοίγει την κλειδωμένη πόρτα.

Όπως και στις προηγούμενες περιπτώσεις έτσι και εδώ πρέπει να υπάρχει ένα ελάχιστο κατώφλι για τον έλεγχο της περιστροφής του κόμβου και τρεις καταστάσεις για το εάν είναι «κλειστός», «ανοιχτός» ή αν η κατάσταση του είναι απροσδιόριστη. Για αυτόν τον λόγο επαναχρησιμοποιούνται οι μεταβλητές που χρησιμοποιήσαμε για τις συναρτήσεις για items που πραγματοποιούν περιστροφή.

```
public static float MIN_FLOAT_ROTATE = 0.2f;
public static int STATE_OPEN_ROTATED = 1;
public static int STATE_CLOSED_ROTATED = -1;
public static int STATE_UNDEFINED_ROTATED = 0;
```

Με το ίδιο σκεπτικό επαναχρησιμοποιείται και η συνάρτηση isEqual όπως ακριβώς αυτή ορίστηκε παραπάνω στις συναρτήσεις για items που πραγματοποιούν μετατόπιση για να βρούμε την θέση του κόμβου σε σχέση και με το ελάχιστο κατώφλι που ορίσαμε.

```
public boolean isEqual(float x0, float y0, float z0, float x1, float y1, float z1) {
        return Math.abs(x1 - x0) < MIN_FLOAT &&
        Math.abs(y1 - y0) < MIN_FLOAT &&
        Math.abs(z1 - z0) < MIN_FLOAT;
}
```

Επόμενο βήμα είναι ο προσδιορισμός της κατάστασης του κόμβου του item. Για αυτό ορίζουμε την συνάρτηση state_nodes_rotated() η οποία δέχεται ως ορίσματα ένα σύμβολο που έχουμε ορίσει στον κόμβο για τον οποίο ενδιαφερόμαστε για τους λόγους που αναλύσαμε παραπάνω, και τις αρχικές και τελικές συντεταγμένες του όπως αυτές θα πρέπει να είναι.

```
public int state_nodes_rotated(String symbolName, float xo, float yo, float zo, float xc, float yc, float zc){

        // get the location symbol instances for the specified item...
        Vector<Vector<String>> locations = getKB().get(symbolName);

        // get the second argument of the first symbol instance found...
        String location = locations.get(0).get(1);

        // the argument is of the form "x,y,z", parse it to get...
        // individual coordinates...
        String[] coords = location.split(",");
        float x = Float.valueOf(coords[0]);
        float y = Float.valueOf(coords[1]);
        float z = Float.valueOf(coords[2]);

        // compare...
```

```
        if (isEqual(x, y, z, xo, yo, zo)) {
                return STATE_OPEN_ROTATED;
        }
        else if (isEqual(x, y, z, xc, yc, zc)) {
                return STATE_CLOSED_ROTATED;
        }
        else {
                return STATE_UNDEFINED_ROTATED;
        }
}
```

Στη συνέχεια για την ολοκλήρωση της ενέργειας είναι απαραίτητος ο ορισμός της συνάρτησης turn_nodes ώστε ο housekeeper να εκτελέσει την ενέργεια της περιστροφής πάνω στον κόμβο του item. Ορίσματα της συνάρτησης turn_nodes() είναι το όνομα του item, το όνομα του accesspoint στο οποίο θα κάνει attach ο housekeeper, το όνομα της συνάρτησης την οποία θα εκτελέσει ο housekeeper, τους άξονες περιστροφής του κόμβου του item και την γωνία περιστροφής σε ακτίνια. Ο housekeeper κάνει attach τον hand effector του στο accespoint του item, στη συνέχεια επιλέγει την συνάρτηση που ορίσαμε για να εκτελέσει την περιστροφή, την εκτελεί και κάνει detach. Η συνάρτηση turn() είναι boolean.

```
public boolean turn_nodes (String itemName, String accessPointName, String functionName, float xa,
float ya, float za, float angle) {
        if (!act("attach leftHand " + itemName + " " + accessPointName)) {
                System.out.println("ERROR: Could not attach leftHand effector to " +
                accessPointName + " accesspoint!");
                return false;
        }
        if (!act("select leftHand " +  " use " + functionName)) {
                System.out.println("ERROR: Could not select " + functionName + " function for turn
                action!");
                return false;
        }
        if (!act("act leftHand "  + " use " + xa + "," + ya + "," + za + ","+ angle)) {
                System.out.println("ERROR: Could not turn " + itemName + "!");
                return false;
        }
        if (!act("attach leftHand " )) {
                System.out.println("ERROR: Could not dettach leftHand effector!");
                return false;
        }
        return true;
}
```

Τελευταίο βήμα αποτελεί ο ορισμός των συναρτήσεων ensureOpen_nodes_rotated και ensureClosed_nodes_rotated όπου υπολογίζονται οι συντεταγμένες του κόμβου του item και συγκρίνονται με τις αρχικές και τελικές συντεταγμένες του. Ορίσματα και των δύο συναρτήσεων είναι το όνομα του item, το σύμβολο που έχουμε ορίσει στον κόμβο, το accesspoint του και το όνομα της function του και τέλος οι αρχικές ή οι τελικές του συντεταγμένες αντίστοιχα, και οι άξονες περιστροφής του και η

γωνία περιστροφής σε ακτίνια. Στην ensureClosed_nodes_rotated οι συντεταγμένες του κόμβου του item συγκρίνονται με τις τελικές συντεταγμένες του με την βοήθεια της isEqual() και σε περίπτωση όπου είναι ίσες τότε καλείται η turn_nodes() για να περιστρέψει τον κόμβο στις αρχικές του συντεταγμένες. Αντίθετα στην ensureClosed_nodes_rotated οι συντεταγμένες του κόμβου του item συγκρίνονται με τις αρχικές συντεταγμένες του με την βοήθεια της isEqual() και σε περίπτωση όπου είναι ίσες τότε καλείται η turn_nodes() για να περιστρέψει τον κόμβο του item στις τελικές του συντεταγμένες.

```
public void ensureOpen_nodes_rotated(String itemName, String symbolName, String accessPointName, String functionName, float xo, float yo, float zo, float xc, float yc, float zc, float xa, float ya, float za,float angle) {

	// get the location symbol instances for the specified item...
	Vector<Vector<String>> locations = getKB().get(symbolName);

	// get the second argument of the first symbol instance found...
	String location = locations.get(0).get(1);

	// the argument is of the form "x,y,z", parse it to get...
	// individual coordinates...
	String[] coords = location.split(",");
	float x = Float.valueOf(coords[0]);
	float y = Float.valueOf(coords[1]);
	float z = Float.valueOf(coords[2]);

	double angle0= (Math.atan2(y - yc, x - xc));
	System.out.println("Angle:  " + angle0 + "!");

	// compare...
	if (isEqual(x, y, z, xo, yo, zo)) {
		// the specified item is already in the "open" state, do nothing...
	}
	else if (isEqual(x, y, z, xc, yc, zc)) {
		turn_nodes(itemName, accessPointName, functionName, xa, ya, za, angle);
	}
	else {
		// the specified item is in an undefined state, do nothing...
	}
}
```

```
public void ensureClosed_nodes_rotated(String itemName, String symbolName, String accessPointName, String functionName, float xo, float yo, float zo, float xc, float yc, float zc, float xa, float ya, float za,float angle) {

	// get the location symbol instances for the specified item...
	Vector<Vector<String>> locations = getKB().get(symbolName);
```

```
        // get the second argument of the first symbol instance found...
        String location = locations.get(0).get(1);

        // the argument is of the form "x,y,z", parse it to get...
        // individual coordinates...
        String[] coords = location.split(",");
        float x = Float.valueOf(coords[0]);
        float y = Float.valueOf(coords[1]);
        float z = Float.valueOf(coords[2]);

        double angle0= (Math.atan2(y - yo, x - xo));
        System.out.println("Angle:  " + angle0 + "!");

        // compare...
        if (isEqual(x, y, z, xo, yo, zo)) {
                turn_nodes(itemName, accessPointName, functionName, xa, ya, za, angle);
        }
        else if (isEqual(x, y, z, xc, yc, zc)) {
                // the specified item is already in the "closed" state, do nothing...
        }
        else {
                // the specified item is in an undefined state, do nothing...
        }
}
```

Παράδειγμα εφαρμογής των παραπάνω αποτελεί το παρακάτω παράδειγμα όπου ο housekeeper ελέγχει εάν υπάρχουν στον κόσμο τα items table5 και key και ελέγχει εάν ο SARA agent έχει περιστρέψει τον κόμβο του συρταριού του τραπεζιού ελέγχοντας το σύμβολο table5_symbol_rotation που έχουμε ορίσει σε αυτό. Εάν αυτό περιστραφεί έτσι ώστε να είναι ανοιχτό ο housekeeper εμφανίζει το κλειδί που ανοίγει την πόρτα.

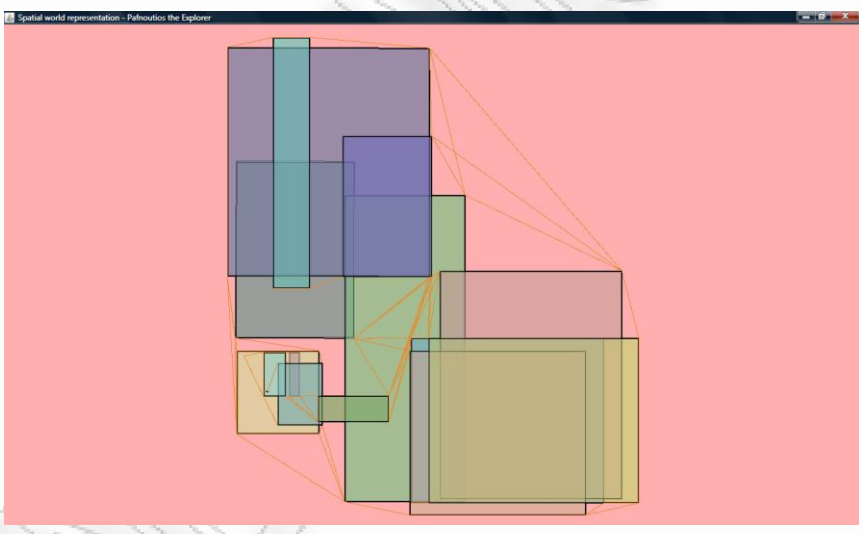
```
if (getKB().get("item", "table5").size()!=0 && getKB().get("item", "key").size()!=0){

        if (state_nodes_rotated("table5_symbol_rotation", 1, 0, 0 , 0, 0,1) ==
        STATE_CLOSED_ROTATED) {
                frame.message("table5_symbol_rotation: Run STATE_CLOSED!");
        }else if (state_nodes_rotated("table5_symbol_rotation",  1, 0, 0 , 0, 0,1) ==
        STATE_OPEN_ROTATED) {
                frame.message("table5_symbol_rotation: Run STATE_OPEN!");
                ensureOpen("key", "key_access_p", "translate_key",0.3f,0,1, 17.8f, 0.5f, -11.5f);
                frame.message("key: Run STATE_OPEN!");
        }else  if (state_nodes_rotated("table5_symbol_rotation",  1, 0, 0 , 0, 0,1) ==
        STATE_UNDEFINED) {
                frame.message("table5_symbol_rotation: Run STATE_UNDEFINED!");
        }else{
```

```
                //do nothing
        }
}
```

## 7.3 SARA Agent

Ο SARA agent είναι ο πράκτορας παίκτης του κόσμου μας ο οποίος δεν γνωρίζει τίποτα για αυτόν. Έχει σώμα αντίθετα με τον housekeeper και για να αποκτήσει οποιαδήποτε γνώση για τον κόσμο θα πρέπει να «δει» πρώτα τα items που τον αποτελούν. Έχει όμως στη διάθεση του ένα σύνολο διαδικασιών pathfinding, wandering, χωρικής αναπαράσταση, ελέγχου της βάσης γνώσης και εξαγωγής συμπερασμάτων από αυτή καθώς και άλλων πολύπλοκων διαδικασιών ώστε να επιτελέσει αυτό τον σκοπό. Για την δημιουργία του δικού μας SARA agent βασιστήκαμε στα παραδείγματα SARA πρακτόρων οι οποίοι είναι υλοποημένοι με βάση τη βιβλιοθήκη της SARA, επεκτείνοντας τους.

Μελετώντας τα παραδείγματα SARA πρακτόρων της βιβλιοθήκης της SARA και εφαρμόζοντας τα στον δικό μας κόσμο και ειδικότερα το SampleAgent6.java στο οποίο ένας πράκτορας περιπλανιέται στον χώρο, παρατηρήθηκε ότι κατά η αναπαράσταση των items που περιέχονται στη βάση γνώσης του πράκτορα και που αναπαρίστόνται στο spatial representation window δεν ήταν η αναμενόμενη. Συγκεκριμένα η αναπαράσταση αφορούσε γεωμετρικά αντικείμενα τα οποία επικαλύπτονταν, χωρίς να έχουν κάποια οπτική αντιστοιχία με τον πραγματικό κόσμο, ενώ ταυτόχρονα η βάση γνώσης δεν ήταν η αναμενόμενη, καθώς ενώ ο πράκτορας μετακινούνταν και έκανε sense μπροστά από κάποιο item αυτό δεν προστίθονταν σε αυτή.




**Λανθασμένη χωρική αναπαράσταση της βάσης γνώσης**

Με γνώμωνα ότι τα παραδείγματα λειτουργούσαν κανονικά για τους κόσμους των παραδειγμάτων και ότι ο ίδιος SARA agent δεν είχε πρόβλημα λειτουργικότητας σε αυτούς, έπρεπε να αναζητηθεί το πρόβλημα στο κόσμο μας και ιδιαίτερα στην αποικόνιση του. Με προσεκτικότερη παρατήρηση, ανακαλύφθηκε ότι κατά την αναπαράσταση του virtual model στη verl, ο κόμβος scale δεν γίνονταν αντιληπτός από τον πράκτορα. Έτσι ενώ οπτικά ως παρατηρητές βλέπουμε τον κόσμο σύμφωνα με τις καινούριες διαστάσεις των items που ορίστηκαν από το scale, ο πράκτορας εξακολουθεί να βλέπει τα items με τις αρχικές τους διαστάσεις. Για να επιλυθεί το πρόβλημα θα έπρεπε να αφαιρεθούν όλοι οι κόμβοι scale από την virtual representation και να ενσωματωθούν στα vrml αρχεία των items αλλά και πάλι όχι σε οποιοδήποτε σημείο αλλά κρατώντας μια συγκεκριμένη δομή τύπου:

```
DEF item Transform {
        children[
                Transform{
                        scale x y z
                                children[
                                …….
```

Έτσι κάθε item χρειάζεται να μετασχηματιστεί , τόσο στο virtual model όσο και στο .wrl αρχείο του σύμφωνα με τα παρακάτω, όπου αριστερά αποικονίζεται η παλία υλοποίηση και δεξιά η ορθή καινούρια υλοποίηση:

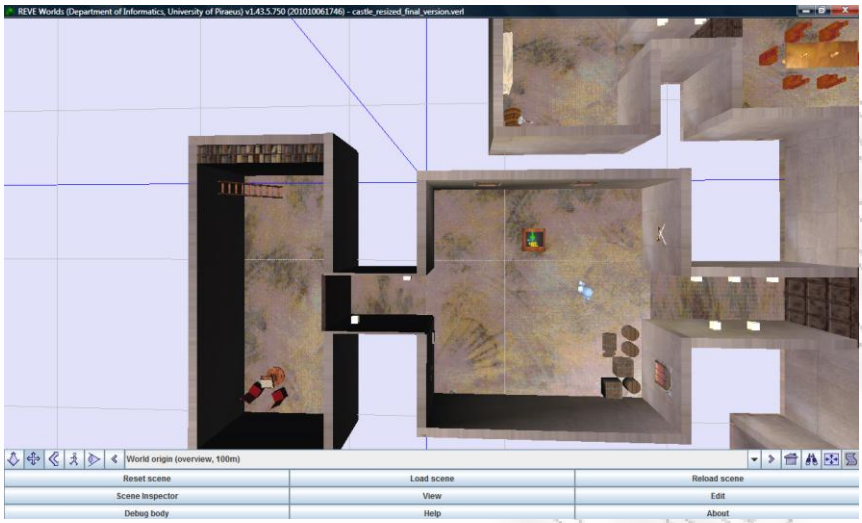
```
<item name="RightWallR1_1">
        <virtualModel source="RightWallR1_1">
          <transform translation="70, 0, 70"
          rotation="0 1 0 1.57" scale="1 0.5 1" />
        </virtualModel>
</item>
```

```
<item name="RightWallR1_1">
        <virtualModel source="RightWallR1_1">
          <transform translation="13.5, 0, 14"
            rotation="0 1 0 1.57"/>
        </virtualModel>
</item>
```

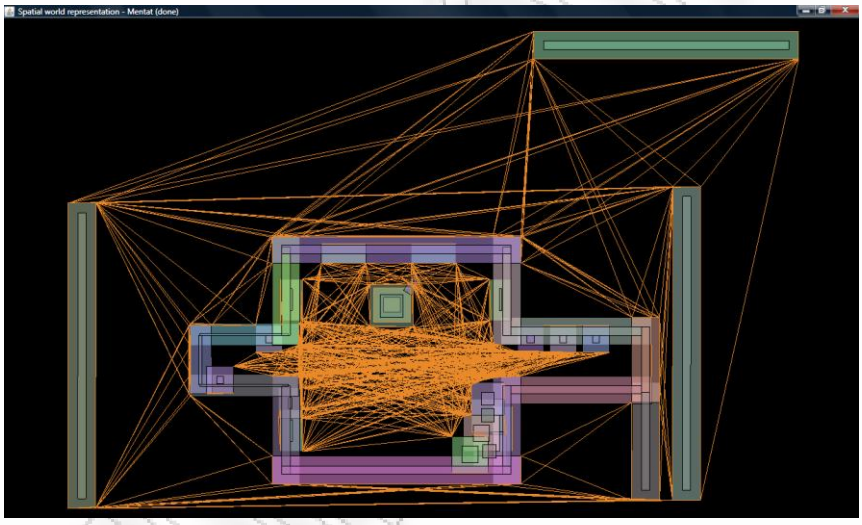
```
DEF RightWallR1_1 Transform {
        translation 70 5 -15
        rotation 0 1 0 1.57
        children [
                Shape {
    ……
    ……
    ……
                }
        ]
}
```

```
DEF RightWallR1_1 Transform {
        children [
                Transform{
                        scale 0.18 0.15 0.5
                        children [
                                Shape{
    ……
    ……
    ……
                                }
                        ]
                }
        ]
}
```

Μετά την εφαρμογή των παραπάνω αλλαγών σε όλα τα items του κόσμου, ο πράκτορας είναι σε θέση να αντιληφθεί το σωστό μέγεθος τους και να μην έχει πρόβλημα λειτουργικότητας στον κόσμο όπως φαίνεται και στις παρακάτω εικόνες.




**Εικονικός κόσμος**




**Χωρική αναπαράσταση της kb του πράκτορα**

Στο σημείο αυτό αναφέρουμε ότι έγινε και resize εκ νέου στον κόσμο έτσι ώστε αυτός να είναι αντίστοιχος με το μέγεθος του πράκτορα.

Αφού λύσαμε και το πρόβλημα της αναπαράστασης της kb, έπρεπε να υλοποιηθεί ένας SARA agent ειδικά για τις ανάγκες του παρόντος εικονικού κόσμου. Οι πράκτορες βασίζονται στην κλάση BaseMotileAgent η οποία προσφέρει μία έκδοση ενός SARA agent με ικανότητα ρεαλιστικής κίνησης προς μία συγκεκριμένη τοποθεσία, pathfinding προς μία συγκεκριμένη τοποθεσία η οποία υπολογίζεται με βάση τα items κλάσης "real" του κόσμου μας, εκτέλεση συγκεκριμένου path, wandering και moveTo.

```
public class sara_castle_version extends BaseMotileAgent implements MotileAgentListener
{…….
…….
}
```

Η λογική της λειτουργίας του housekeeper είναι η εξής: Ο πράκτορας συνδέεται στο port 4444. Κατά την διάρκεια της αρχικοποίησης του, δημιουργείται το user interface το οποίο αποτελείται από ένα spatial representation window, στο οποίο σε συνδυασμό με το navigation mesh απεικονίζονται τα items που

υπάρχουν στην βάση γνώσης του πράκτορα και οι διαδρομές που αυτός υπολογίζει, και γίνεται attach ο motor effector του πράκτορα στο motor accessspoint και επιλέγεται για translation αλλά και για rotation έτσι ώστε ο πράκτορας να μπορεί να μετακινηθεί αποτελεσματικά στον χώρο χωρίς να δημιουργείται πρόβλημα λειτουργικότητας. Σε περίπτωση όπου για κάποιο λόγο δεν γίνει η αρχικοποίηση, ο χρήστης λαμβάνει σχετικό μήνυμα αποτυχίας.

Για την υλοποίηση της παραπάνω διαδικασίας χρησιμοποιούμε κάποιες συναρτήσεις από της βιβλιοθήκη της SARA. Η πρώτη από αυτές είναι την BaseAgent.Status(), οποία επιστρέφει μια enum type μεταβλητή η οποία εκφράζει την κατάσταση στην οποία βρίσκεται ο πράκτορας. Ένας SARA πράκτορας μπορεί να βρίσκεται σε μία από τις ακόλουθες καταστάσεις:

- CONNECTED
- CONNECTING
- EXECUTING
- HANDSHAKING
- INITIALIZED
- INITIALIZING
- SENSING
- STALE

Στη συνέχεια χρησιμοποιείται η getName() η οποία επιστρέφει το όνομα του SARA πράκτορα του κόσμου μας ως string.

Η συνάρτηση connect(String address, Int port) είναι τύπου boolean και περιμένει μέχρι είτε να δημιουργηθεί σύνδεση όπου και επιστρέφει true, είτε αυτή να μην γίνει δεκτή είτε να επιστραφεί κάποιο λάθος στην σύνδεση.

Η act() είναι επίσης τύπου boolean και εκτελεί μία εντολή την οποία δέχεται ως παράμετρο.

Οι συναρτήσεις sense() και update() χρησιμοποιούνται για να λάβει πληροφορία ο πράκτορας για τα items τα οποία βρίσκονται στο οπτικό του πεδίο (field of sense) και να ενημερώσει τη βάση γνώσης του με τις καινούριες πληροφορίες.

Τέλος η frame.repaint() χρησιμοποιείται για να ενημερωθεί το user interface με τα καινούρια δεδομένα τα οποία βρίσκονται στη βάση γνώσης του πράκτορα μας.

```
/** The agent's GUI. */
public SaraAgent_ver2Frame frame;

/*** Initializes the agent's name, connects to a local AIPd on port 4444, attaches the "motor" effector to
* the "motorAccessPoint" accesspoint, selects the "translate" function for the "move" action and the
* "rotate" function for the "turn" action, performs a sense operation, updates the item map and updates the
* GUI.
* @param name the agent's name, as a {@link String}
*/
public sara_castle_version(String name) {

        super(name);

        if (status.equals(Status.INITIALIZED)) {

                System.out.println("[" + getName() + "] Initialized.");

                addMotileAgentListener(this);
```

```
                System.out.print("[" + getName() + "] Creating spatial representation window... ");
                frame = new SaraAgent_ver2Frame("Spatial world representation - " + getName(),
                getNavMesh());
                frame.setSize(640, 480);
                frame.validate();
                frame.setLocationRelativeTo(null);
                frame.setVisible(true);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                System.out.println("Done.");

                System.out.print("[" + getName() + "] Connecting... ");
                try {
                        connect("127.0.0.1", 4444);
                }
                catch (IOException ex) {
                        System.out.println("\n[" + getName() + "] Network error while connecting (" +
                        ex.getMessage() + ")");
                }

                if (status.equals(Status.CONNECTED)) {
                        System.out.println("Done.");
                        System.out.print("[" + getName() + "] Attaching motor effector to motor
                        accesspoint... ");
                        act("attach motor motorAccessPoint");
                        System.out.println("Done.");

                        System.out.print("[" + getName() + "] Selecting translation action... ");
                        act("select motor move translate");
                        System.out.println("Done.");

                        System.out.print("[" + getName() + "] Selecting rotation action... ");
                        act("select motor turn rotate");
                        System.out.println("Done.");

                        sense();
                        update();
                }
        }
        frame.repaint();
}
```

Μετά την αρχικοποίηση του πράκτορα και την σύνδεση του σειρά έχει η δημιουργία της μεθόδου main(String[] args) η οποία δημιουργεί και χρησιμοποιεί τον πράκτορα της κλάσης sara_castle_version χρησιμοποιώντας το public interface της κλάσης. Σε αυτό το σημείο ορίζεται και το όνομα του SARA πράκτορα μας. Ο πράκτορας μας πλέον ονομάζεται Mentat, κατά τους ανθρώπους-υπολογιστές του

μυθιστορήματος Dune του Frank Herbert. Σε περίπτωση όπου η σύνδεση έχει δημιουργηθεί επιτυχώς τότε καλείται η συνάρτηση start().

```
/** Main method, creates and uses an agent of the class using the class' public interface. *
* @param args command-line arguments, not used by this implementation
*/
public static void main(String[] args) {

	sara_castle_version agent = new sara_castle_version("Mentat");

	// check if the agent has been succesfully initialized...

	if (agent.getStatus().equals(Status.CONNECTED)) {

		System.out.println("Created agent " + agent.getName() + ".");
		System.out.print("Initiating agent behaviour... ");
		agent.start();
		agent.frame.setTitle(agent.frame.getTitle() + " (done)");
		System.out.println("Done. Close spatial representation window to terminate
		 application.");


	}
	else {
		// initialization of the agent has failed, for some reason...
		System.out.println("ERROR: Agent initialization error!");
		return;
	}
}
```

Η συνάρτηση start() ουσιαστικά αρχικοποιεί την συμπεριφορά του πράκτορα μας. Μόλις αυτός συνδεθεί στον κόσμο μας, κάνει sense και με βάση την βάση γνώσης του ενημερώνεται το item map και το spatial representation window και ο πράκτορας αρχίζει να περιπλανιέται στον κόσμο μας με την κλήση της συνάρτησης startWandering().

```
/** Initializes execution of the agent's behavior, which consists of wandering to random locations
*around the world and within a specific range beyond the limit of their sensory range. To move to each
*location, the agent will first turn so that is faces it and then start moving until it reaches it.
*/
public void start() {

	System.out.print("Sensing... ");
	sense();
	System.out.println("Done.");
```

```
        System.out.println("Dumping KB...");
        System.out.print(getKB().getReadableForm());
        System.out.println("Done.");

        System.out.print("Updating item map... ");
        update();
        System.out.println("Done.");

        System.out.println("Getting spatial information...");
        System.out.println("Number of rectangles: " + getItemMap().size());
        float[] extent = getItemMap().getExtent();
        System.out.println("Known world extent: (" + extent[0] + ", " + extent[1] + ") - (" + extent[2] +
         ", " + extent[3] + ")");
        System.out.println("Done.");

        System.out.print("Updating spatial representation window... ");
        frame.repaint();
        System.out.println("Done.");

        System.out.print("Acting... ");
        wander=1;
        startWandering();
        System.out.println("[" + getStatus() + "] Status.");
        System.out.println("Done.");
}
```

Σε αυτό το σημείο θα αναλυθούν κάποια πράγματα σχετικά με τον τρόπο που δρα και κινείται ένας SARA agent. Με την κλήση της συνάρτησης startWandering ο πράκτορας αρχίζει στηριζόμενος στη βάση γνώσης του να εκτελεί τυχαία κίνηση στον χώρο-κόσμο. Ο πράκτορας συνεχίζει το wandering μέχρι να κάνει για πρώτη φορά sense σε κάποιους διακόπτες ή items με τα οποία μπορεί να αλληλεπιδράσει και να αλλάξει τον αρχικό κόσμο. Ο λόγος που η υλοποίηση έγινε με βάση την πρώτη μόνο φορά που ο πράκτορας βλέπει τα items είναι λόγο του μεγάλου μεγέθους του κόσμου και του χρόνου τον οποίο χρειάζεται ο πράκτορας για να τον εξερευνήσει. Έτσι δρα σε αυτά μόνο μία φορά, ανοίγοντας μόνο ουσιαστικά ένα δωμάτιο στον κόσμο χωρίς μετά να κλείνει την είσοδό του. Η διαδικασία του ανοίγματος και κλεισίματος ανάλογα με την κατάσταση τους των items δεν θα έβλαπτε σε τίποτα την τωρινή λειτουργικότητα του, και ο λόγος τις μονόδρομης υλοποίησης είναι για λόγους ταχύτερης παρουσίασης (presentation) των ικανοτήτων του πράκτορα. Για να κάνει wandering ο πράκτορας καλεί τα ακόλουθα threads:

- translationCompleted
- rotationCompleted
- segmentStarted
- segmentCompleted
- pathCompleted

Κάθε φορά ο SARA agent δημιουργεί ένα μονοπάτι στο μυαλό του, από την θέση που βρίσκεται στην θέση όπου επέλεξε να πάει, το οποίο αποτελείται από επιμέρους translations και rotations. Κάθε φορά όμως που ολοκληρώνεται ένα από αυτά τα rotations ή translations, ο πράκτορας θα πρέπει να ελέγξει εάν στο field of sense του βρίσκεται κάποιο item με το οποίο μπορεί να αλληλεπιδράσει. Ο συγκεκριμένος τρόπος δράσης του SARA agent είναι πολύ αποδοτικός, καθώς κάποιο path που θα μπορούσε να επιλέξει

για εκτέλεση ο πράκτορας μπορεί να ήταν πολύ μεγάλο ή σε κάθε περίπτωση μπορεί να περνούσε πολύ κοντά από το item που αναζητούσε και να μην το έβλεπε γιατί δεν είχε ολοκληρώσει ολόκληρο το path. Σε κάθε περίπτωση μια τέτοια υλοποίηση θα είχε ελάχιστες πιθανότητες επιτυχίας σε αντίθεση με τώρα. Έτσι τα παραπάνω threads ανακατασκευάστηκαν έτσι ώστε όταν ολοκληρώνεται ένα rotation ή ένα translation να καλείται η συνάρτηση dododo() η οποία αποτελείται από τις ενέργειες που πρέπει να κάνει ο Mentat στον τρέχων κόσμο όπως αυτή αναλύθηκε στο walkthrough.

Με την κλήση της dododo() ο πράκτορας ελέγχει εάν στη βάση γνώσης του έχει προστεθεί κάποιο item που τον ενδιαφέρει. Σε αυτή την περίπτωση θα πρέπει να πάει κοντά του και να αλληλεπιδράσει με αυτό. Όμως η λογική του SARA agent είναι τέτοια, όπου ένας πράκτορας πρέπει να έχει τελειώσει μία ενέργεια για να εκτελέσει μία άλλη καθώς τα threads καλούνται από πολλές διαδικασίες ταυτόχρονα και σε τέτοια περίπτωση το σύστημα οδηγείται σε exception και τερματισμό. Για να αναλύσουμε αυτή την συμπεριφορά του πράκτορα θα αναφέρουμε ένα παράδειγμα. Έστω ένας πράκτορας έχει συνδεθεί στον κόσμο μας και κάνει wandering. Κάποια στιγμή κάνει sense ένα item το οποίο τον ενδιαφέρει και πρέπει να δράσει σε αυτό. Για να δράσει ένα SARA πράκτορας σε ένα item καλό θα είναι να πάει κοντά του έτσι ώστε να εξασφαλίσουμε ότι θα «δεί» την καινούρια κατάσταση του μετά την δράση του και να ενημερωθεί για αυτήν. Ένας λόγος για τον οποίο θα μπορούσε να μην δεί την καινούρια κατάσταση του item είναι λόγω της οπτικής γωνίας από την οποία έδρασε σε αυτό και της συμπεριφοράς του item μετά την δράση του πράκτορα, πρόβλημα το οποίο θα μπορούσε να λυθεί και με πυκνότερο ray casting από πλευράς του πράκτορα και μεγαλύτερο εύρος της οπτικής του, το οποίο όμως δεν μπορεί να εξασφαλίσει σε κάθε περίπτωση ότι θα ενημερωθεί για την καινούρια κατάσταση. Σε περίπτωση όπου η βάση γνώσης δεν ενημερωθεί για τα καινούρια δεδομένα, τότε δημιουργείται πρόβλημα λειτουργικότητας στον κόσμο, καθώς ο πράκτορας θα εκτελεί συνεχώς μια δράση σε ένα item το οποίο έχει αλλάξει κατάσταση για τον κόσμο όχι όμως και για τον πράκτορα. Για να μετακινηθεί ο πράκτορας κοντά στο item που θέλει, καλούμε τις συναρτήσεις Pathfind(x,y) και StartPathExcecution(). Η πρώτη υπολογίζει μια διαδρομή προς ένα συγκεκριμένο προορισμό, με βάση τα items που υπάρχουν στη βάση γνώσης του πράκτορα, και η δεύτερη εκτελεί αυτή την διαδρομή. Η StartPathExcecution() όμως καλεί με τη σειρά της τα threads που προαναφέραμε. Η έναρξη δηλαδή της StartpathExcecution() δεν σταματάει αυτόματα την startWandering() αλλά και οι δύο καλούν τα ίδια threads με αποτέλεσμα να οδηγούμαστε σε exception. Επιπρόσθετα σε περίπτωση όπου π.χ. στη συνάρτηση dododo() είχαμε μια ακολουθία της μορφής startWandering(), stopWandering(),Pathfind(x,y), StartPathExcecution(),startWandering() θα οδηγούμασταν και πάλι σε exception καθώς κάθε μια διαδικασία καλεί τα threads τα οποία με τη σειρά τους καλούν την διαδικασία με αποτέλεσμα να ζητείται από τον πράκτορα να κάνει πράγματα ταυτόχρονα. Για την επίλυση του προβλήματος δημιουργήθηκε ένα απλό finite state machine με τις καταστάσεις στις οποίες μπορεί να βρίσκεται ένας πράκτορας. Πιο συγκεκριμένα ένας πράκτορας μπορεί να κάνει wandering, να κάνει pathExcecution, να έχει ολοκληρώσει το pathExecution και να έχει βρει την έξοδο όπου είναι και η τελική κατάσταση του πράκτορα στην οποία βρίσκεται μόνο μία φορά στο τέλος όλης της διαδικασίας.

Έτσι η καινούρια λογική του πράκτορα είναι η εξής: Ο πράκτορας αρχικοποιείται και αρχίζει να κάνει wandering. Σε περίπτωση όπου κάνει sense κάποιο item στο οποίο μπορεί να αλληλεπιδράσει κάνει stopWandering ενεργεί σε αυτό και αφού κάνει pathFind αρχίζει το PathExecution. Σε κάθε περίπτωση ελέγχουμε εάν έχει ολοκληρώσει αυτό το path. Μόλις αυτό ολοκληρωθεί ο πράκτορας δεν κάνει ούτε wandering ούτε pathExecution αλλά υπολογίζει την γωνία μεταξύ αυτού και του item και γυρίζει προς αυτό για να κάνει sense την καινούρια του κατάσταση. Μόλις ενημερωθεί η βάση γνώσης του ξαναρχίζει το wandering.

Για την υλοποίηση αυτής της λογικής χρειάστηκε να τροποποιηθούν τα threads όπως αυτά προσφέρονται στα παραδείγματα της βιβλιοθήκης της SARA σύμφωνα με τα ακόλουθα και δημιουργήθηκε μία ακόμη συνάρτηση η restartWandering() η οποία διαχειρίζεται ουσιαστικά το finite state machine.

```
* Performs a sensory operation, updates the item map and navigation mesh, and updates the GUI.
* @see MotileAgentListener#translationCompleted(reve.sara.agent.MotileAgentEvent)
* @param ev
*/
@Override
```

```
public void translationCompleted(MotileAgentEvent ev) {
	System.out.println("[" + getName() + "] Translation completed.");
	sense();
	update();
	frame.repaint();
	if (wander==1){
		dododo();
	}
}
/** Performs a sensory operation, updates the item map and navigation mesh, and updates the GUI.
* @see MotileAgentListener#rotationCompleted(reve.sara.agent.MotileAgentEvent)
* @param ev
*/
@Override
public void rotationCompleted(MotileAgentEvent ev) {
	System.out.println("[" + getName() + "] Rotation completed.");
	sense();
	update();
	frame.repaint();
	if (wander==1){
		dododo();
	}
	System.out.println("[" + getStatus() + "] Status.");
	if (wander==0){
		wander=1;
		restartWandering();

	}
}

/** Performs a sensory operation, updates the item map and navigation mesh, and updates the GUI.
* @see MotileAgentListener#segmentStarted(reve.sara.agent.MotileAgentEvent)
* @param ev
*/
@Override
public void segmentStarted(MotileAgentEvent ev) {
	System.out.println("[" + getName() + "] Segment started.");
	sense();
	update();
	frame.repaint();
	System.out.println("[" + getStatus() + "] Status.");
}

/**
* @see MotileAgentListener#segmentCompleted(reve.sara.agent.MotileAgentEvent)
```

```
* @param ev
*/
@Override
public void segmentCompleted(MotileAgentEvent ev) {
	System.out.println("[" + getName() + "] Segment completed.");
	System.out.println("[" + getStatus() + "] Status.");
}

/**
* @see MotileAgentListener#pathCompleted(reve.sara.agent.MotileAgentEvent)
* @param ev
*/
@Override
public void pathCompleted(MotileAgentEvent ev) {
	System.out.println("[" + getName() + "] Path completed.");
	System.out.println("[" + getStatus() + "] Status.");

	if (wander==2)  {
		System.out.println("StartWanderinggggggggggggg");
		wander=0;
		restartWandering();
	}
	if (wander==0)  {
		System.out.println("StartWanderinggggggggggggg");
		wander=1;
		restartWandering();
	}
	if (wander==2)  {
		restartWandering();
	}
	if (wander==4){
		System.out.println("I am free!!!!!!!!!!!.");
		hangup();
	}
System.out.println("[" + wander + "] wander.");
}
```

```
public void restartWandering(){
	if (wander==0){
		rotation=getAngleTo(Dx, Dz);
		System.out.println(rotation+"RRRotation");
		act("act motor turn "+ rotation);
		System.out.println("turn!!!!!!!!!!!.");
	}
	if (wander==1){
```

```
                startWandering();
                System.out.println("Done!!!!!!!!!!!!.");
        }
        System.out.println("[" + wander + "] wander.");
        sense();
        update();
        frame.repaint();
}
```

Κατά τη διάρκεια αυτής της υλοποίησης παρουσιάστηκε πρόβλημα κατά την διαδικασία αλλαγής της κατάστασης του πράκτορα, δηλαδή από τη στιγμή που τερματίζονταν κάποιο thread μετά από την stopWandring() τότε δεν ήταν εφικτό να ξαναγίνει έναρξή του με κλήση των συναρτήσεων StartPAthExecution, moveTo ή startWandering() καθώς σε αυτές οι λειτουργίες μπορούσαν να εκτελεστούν σε περίπτωση όπου το thread ήταν alive κάτι που δεν ίσχυε από την στιγμή που γίνονταν stopWandering(). Το πρόβλημα αυτό διορθώθηκε από τον κ. Γιώργο Αναστασσάκη με αλλαγή της βιβλιοθήκης της SARA.

Όσων αφορά την λειτουργικότητα του πράκτορα σε σχέση με τα items του κόσμου και την επίδραση του σε αυτά δημιουργήθηκαν οι ακόλουθες συναρτήσεις οι οποίες θα αναλυθούν παρακάτω:

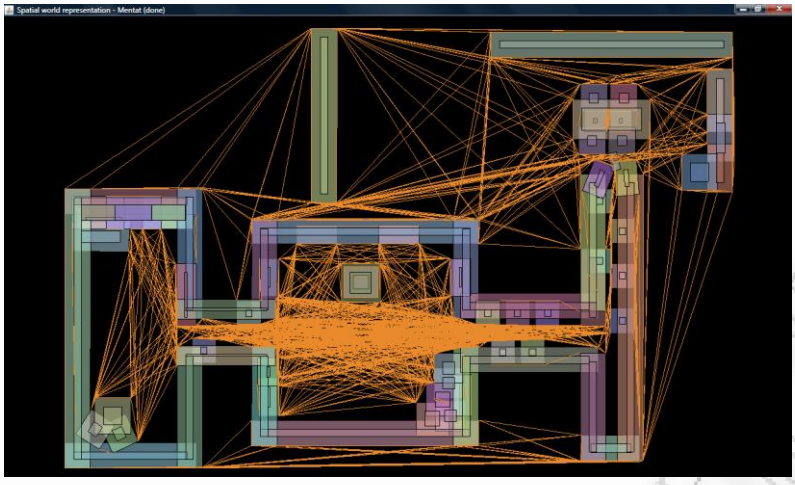
- Συναρτήσεις για items τα οποία πραγματοποιούν κατά την κίνηση τους translation
- Συναρτήσεις για items τα οποία πραγματοποιούν κατά την κίνηση τους rotation
- Συναρτήσεις για κόμβους items τα οποία πραγματοποιούν κατά την κίνηση τους translation
- Συναρτήσεις για κόμβους items τα οποία πραγματοποιούν κατά την κίνηση τους rotation
- Συναρτήσεις για την σύγκριση συμβόλων

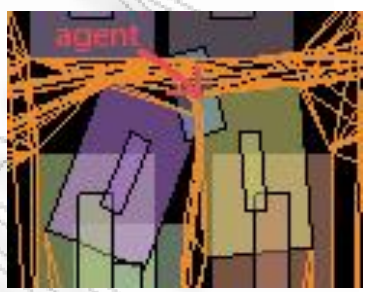
Οι τέσσερις πρώτες από αυτές τις συναρτήσεις είναι ουσιαστικά ίδιες με αυτές του housekeeper με αντιμετώπιση όλων των προβλημάτων ακριβώς όπως και στην περίπτωση του housekeeper με μικρές μόνο διαφορές για την προσαρμογή στις ανάγκες του SARA agent.

Η συνάρτηση start() όπου και υλοποιεί την συμπεριφορά του πράκτορα για τον κόσμο μας είναι ουσιαστικά μια συλλογή των παραπάνω συναρτήσεων σε συνδυασμό με την λογική wandering του πράκτορα όπως αυτή αναλύθηκε παραπάνω.

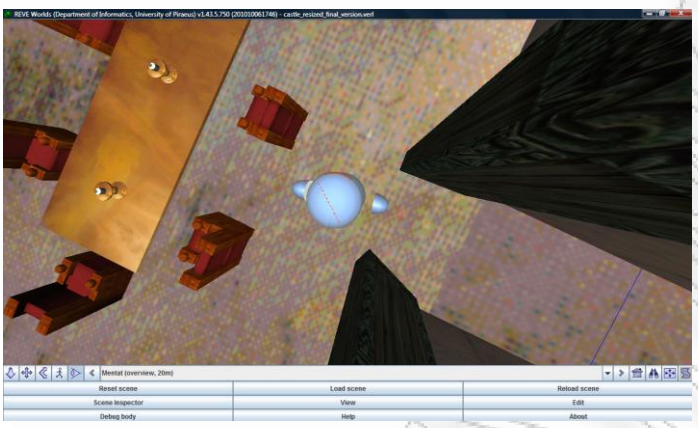
Υλοποιώντας όλη την συμπεριφορά του πράκτορα και δοκιμάζοντας τον στον εικονικό μας κόσμο, παρατηρήθηκε ότι από ένα σημείο και μετά ο πράκτορας φαίνεται να ενεργεί αρκετά αργά μετά την ολοκλήρωση μιας διαδικασίας translation ή rotation. Ο λόγος για αυτή την συμπεριφορά του πράκτορα είναι ότι κάθε φορά που γίνεται update() και frame.repaint(), γίνεται σε ολόκληρη τη βάση γνώσης που όπως έχει διαμορφωθεί μέχρι στιγμής. Έτσι μεγαλώνοντας η βάση γνώσης «καθυστερεί» και ο πράκτορας στην ενημέρωση της. Για την αντιμετώπιση του προβλήματος περιορίστηκαν οι δύο αυτές συναρτήσεις στο ελάχιστο δυνατό και ταυτόχρονα υλοποιήθηκε μια καινούρια δεύτερη έκδοση του κόσμου, αποκλειστικά για λόγους παρουσίασης, όπου έχουν αφαιρεθεί όλα τα items που δεν έχουν λειτουργικότητα, και ουσιαστικά είναι ένας κόσμος που αποτελείται από δωμάτια, πόρτες και διακόπτες/μηχανισμούς για τον έλεγχο τους.

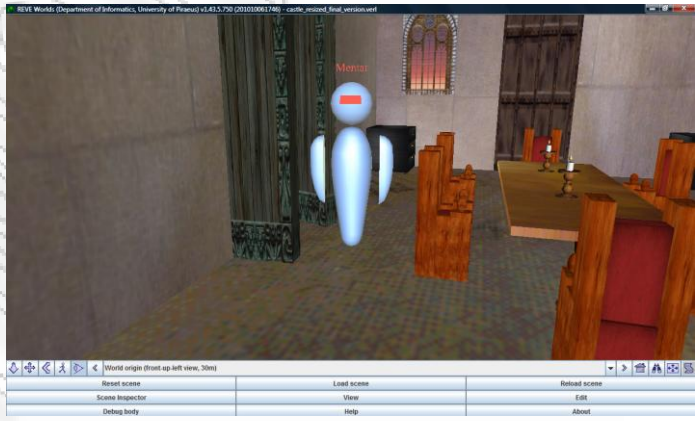
Τέλος ένα σημαντικό πρόβλημα το οποίο ακόμη δεν έχει λυθεί είναι ο εγκλωβισμός πράκτορα κάποιες φορές σε κάποια bounding boxes. Το πρόβλημα παρατηρείται όταν ο πράκτορας ολοκληρώνει ένα segment πολύ κοντά σε ένα item. Τότε αρχίζει να κινείται συνεχώς προς την ίδια θέση με αποτέλεσμα να παραμένει ακίνητος, ή κάνει συνεχώς το ίδιο rotation και έπειτα να μετακινείται προς την ίδια θέση. Το πρόβλημα εντοπίζεται στο εξής, ο πράκτορας έχει προσθέσει κάποια items στη βάση γνώσης του και επιλέγει κάποια διαδρομή. Κατά την ολοκλήρωση της κάνει και πάλι sense. Αν βρίσκεται πολύ κοντά σε κάποιο item και τότε μπορεί να υπολογίσει ότι το bounding box του οριακά εισχώρησε στο bounding box του item με αποτέλεσμα να εγκλωβιστεί σε αυτό και να μην μπορεί να απομακρυνθεί από αυτό.




**Αναπαράσταση βάσης γνώσης εμφάνισης προβλήματος εγκλωβισμού σε bb**





**Πράκτορας εγκλωβισμένος σε bb**

Η συμπεριφορά του πράκτορα κατά την εμφάνιση του παρακάτω προβλήματος όπως καταγράφεται στο reve_worls.log παρουσιάζεται παρακάτω:

20117274224] [TCPIPBodyControllerL1 ] (DBG) run: Received response "act motor move 23.660578,-4.4190526 false 2.0"
[20117274224] [TCPIPBodyControllerL1 ] (DBG) run: Delegating command "act" to managed body controller
[20117274224] [TCPIPBodyControllerL1 ] (DBG) run: Sent command prompt, waiting for response
[20117274224] [TCPIPBodyControllerL1 ] (DBG) run: Received response "sense"
[20117274224] [TCPIPBodyControllerL1 ] (DBG) run: Delegating command "sense" to managed body controller
[20117274224] [TCPIPBodyControllerL1 ] (DBG) run: Sent command prompt, waiting for response
[20117274224] [TCPIPBodyControllerL1 ] (DBG) run: Received response "sense"
[20117274224] [TCPIPBodyControllerL1 ] (DBG) run: Delegating command "sense" to managed body controller
[20117274224] [TCPIPBodyControllerL1 ] (DBG) run: Sent command prompt, waiting for response
[20117274225] [TCPIPBodyControllerL1 ] (DBG) run: Received response "sense"
[20117274225] [TCPIPBodyControllerL1 ] (DBG) run: Delegating command "sense" to managed

body controller
[20117274225] [TCPIPBodyControllerL1 ] (DBG) run: Sent command prompt, waiting for response
[20117274225] [TCPIPBodyControllerL1 ] (DBG) run: Received response "sense"
[20117274225] [TCPIPBodyControllerL1 ] (DBG) run: Delegating command "sense" to managed body controller
[20117274225] [TCPIPBodyControllerL1 ] (DBG) run: Sent command prompt, waiting for response
[20117274225] [TCPIPBodyControllerL1 ] (DBG) run: Received response "sense"
[20117274225] [TCPIPBodyControllerL1 ] (DBG) run: Delegating command "sense" to managed body controller
[20117274225] [TCPIPBodyControllerL1 ] (DBG) run: Sent command prompt, waiting for response
[20117274226] [TCPIPBodyControllerL1 ] (DBG) run: Received response "sense"
[20117274226] [TCPIPBodyControllerL1 ] (DBG) run: Delegating command "sense" to managed body controller
[20117274226] [TCPIPBodyControllerL1 ] (DBG) run: Sent command prompt, waiting for response
[20117274227] [TCPIPBodyControllerL1 ] (DBG) run: Received response "act motor turn -6.278332E-8"
[20117274227] [TCPIPBodyControllerL1 ] (DBG) run: Delegating command "act" to managed body controller
[20117274228] [TCPIPBodyControllerL1 ] (DBG) run: Sent command prompt, waiting for response
[20117274228] [TCPIPBodyControllerL1 ] (DBG) run: Received response "sense"
[20117274228] [TCPIPBodyControllerL1 ] (DBG) run: Delegating command "sense" to managed body controller
[20117274228] [TCPIPBodyControllerL1 ] (DBG) run: Sent command prompt, waiting for response
[20117274228] [TCPIPBodyControllerL1 ] (DBG) run: Received response "sense"
[20117274228] [TCPIPBodyControllerL1 ] (DBG) run: Delegating command "sense" to managed body controller
[20117274229] [TCPIPBodyControllerL1 ] (DBG) run: Sent command prompt, waiting for response
[20117274229] [HousekeeperTCPIPBodyControllerL1] (DBG) run: Received response "sense"
[20117274229] [HousekeeperTCPIPBodyControllerL1] (DBG) run: Delegating command "sense" to managed body controller
[20117274229] [HousekeeperTCPIPBodyControllerL1] (DBG) run: Sent command prompt, waiting for response
[20117274229] [TCPIPBodyControllerL1 ] (DBG) run: Received response "sense"
[20117274229] [TCPIPBodyControllerL1 ] (DBG) run: Delegating command "sense" to managed body controller
[20117274229] [TCPIPBodyControllerL1 ] (DBG) run: Sent command prompt, waiting for response
[20117274229] [TCPIPBodyControllerL1 ] (DBG) run: Received response "sense"
[20117274229] [TCPIPBodyControllerL1 ] (DBG) run: Delegating command "sense" to managed body controller
[20117274229] [TCPIPBodyControllerL1 ] (DBG) run: Sent command prompt, waiting for response
[20117274229] [TCPIPBodyControllerL1 ] (DBG) run: Received response "sense"
[20117274229] [TCPIPBodyControllerL1 ] (DBG) run: Delegating command "sense" to managed body controller
[20117274230] [TCPIPBodyControllerL1 ] (DBG) run: Sent command prompt, waiting for response
[20117274230] [TCPIPBodyControllerL1 ] (DBG) run: Received response "act motor move 23.660578,-4.4190526 false 2.0"

```
[20117274230] [TCPIPBodyControllerL1 ] (DBG) run: Delegating command "act" to managed body controller
[20117274230] [TCPIPBodyControllerL1 ] (DBG) run: Sent command prompt, waiting for response
```

### 7.3.1 Συναρτήσεις για items τα οποία πραγματοποιούν κατά την κίνηση τους translation

Βασική λειτουργία την οποία καλείται να εκτελέσει ο SARA agent στα items είναι η μετατόπιση ολόκληρων items όπως συνέβαινε και στην περίπτωση του housekeeper. Έτσι χρησιμοποιούνται οι αρχικοποιήσεις και οι συναρτήσεις isEqual(), state(), move(), ensureOpen(), ensureClosed() ακριβώς όπως στην περίπτωση του housekeeper και για τους ίδιους ακριβώς λόγους με μόνη διαφορά τον effector που χρησιμοποιείται στην move() καθώς ο SARA agent δεν διαθέτει hand effector. Έτσι έχει χρησιμοποιηθεί ο leftHand effector τυχαία ανάμεσα στους effector του SARA agent.

```
public boolean move(String itemName, String accessPointName, String functionName, float x, float y, float z) {
	if (!act("attach leftHand " + itemName + " " + accessPointName)) {
		System.out.print("ERROR: Could not attach leftHand effector to " + accessPointName
		+ " accesspoint!");
		return false;
	}
	if (!act("select leftHand " +  " move " + functionName)) {
		System.out.print("ERROR: Could not select " + functionName + " function for move
		action!");
		return false;
	}
	if (!act("act leftHand "  + " move " + x + "," + y + "," + z + " true")) {
		System.out.print("ERROR: Could not move " + itemName + "!");
		return false;
	}
	if (!act("attach leftHand " )) {
		System.out.print("ERROR: Could not dettach leftHand effector!");
		return false;
	}
	return true;
}
```

Χαρακτηριστικό παράδειγμα της παραπάνω λειτουργικότητας αποτελεί το παρακάτω παράδειγμα όπου εάν ο SARA agent κάνει sense το item Door1opener τότε ελέγχει την κατάσταση του και εάν αυτό είναι «κλειστό» τότε σταματάει το wandering, δρα πάνω του, πάει κοντά του και ενημερώνει τη βάση γνώσης του.

```
if (getKB().get("item", "Door1opener").size() != 0 ) {
	if (state("Door1opener",  0.4f,1, 10, 0.6f, 1, 10) == STATE_OPEN)  {
		System.out.println("Door1opener: Run STATE_OPEN");
		System.out.println("[" + getPath() + "] Status.");
	}
	else if (state("Door1opener",  0.4f,1, 10, 0.6f, 1, 10) == STATE_CLOSED) {
```

```
                if (wander==1){
                        stopWandering();
                        wander=2;
                        System.out.println("Found Door1opener.");
                        System.out.println("Door1opener: Run State_CLOSED");
                        ensureOpen("Door1opener", "mover", "move", 0, 0, -0.2f, 0.6f, 1, 10 );

                        // get the location symbol instances for the specified item...
                        Vector<Vector<String>> Dlocations = getKB().get("location",
                        "Door1opener");

                        // get the second argument of the first symbol instance found...
                        String Dlocation = Dlocations.get(0).get(1);

                        // the argument is of the form "x,y,z", parse it to get...
                        // individual coordinates...
                        String[] coords = Dlocation.split(",");
                        Dx = Float.valueOf(coords[0]);
                        Dy = Float.valueOf(coords[1]);
                        Dz = Float.valueOf(coords[2]);

                        System.out.println("["+ Dx + Dy + Dz +"] New location.");

                        pathfind(2, 9.5f);
                        startPathExecution();
                        sense();
                        update();
                        frame.repaint();
                }
        }
        else if (state("Door1opener", 0.4f,1, 10, 0.6f, 1, 10) == STATE_UNDEFINED) {
                System.out.println("Door1opener: Run STATE_UNDEFINED");
        }
        else {
                //do nothing
        }
}
```

### 7.3.2 Συναρτήσεις για items τα οποία πραγματοποιούν κατά την κίνηση τους rotation

Επόμενη βασική λειτουργία την οποία καλείται να εκτελέσει ο SARA agent στα items είναι η περιστροφή ολόκληρων items όπως συνέβαινε και στην περίπτωση του housekeeper. Έτσι χρησιμοποιούνται οι αρχικοποιήσεις και οι συναρτήσεις isEqual_ROTATED(), state_ROTATED(), turn(), ensureOpen_ROTATED(), ensureClosed_ROTATED() ακριβώς όπως στην περίπτωση του housekeeper και για τους ίδιους ακριβώς λόγους με μόνη διαφορά τον effector που χρησιμοποιείται στην

turn() καθώς ο SARA agent δεν διαθέτει hand effector. Έτσι έχει χρησιμοποιηθεί ο leftHand effector τυχαία ανάμεσα στους effector του SARA agent.

```
public boolean turn (String itemName, String accessPointName, String functionName, float xa, float ya, float za, float angle) {
	if (!act("attach leftHand " + itemName + " " + accessPointName)) {
		System.out.println("ERROR: Could not attach leftHand effector to " +
		accessPointName + " accesspoint!");
		return false;
	}
	if (!act("select leftHand " +  " turn " + functionName)) {
		System.out.println("ERROR: Could not select " + functionName + " function for turn
		action!");
		return false;
	}
	if (!act("act leftHand "  + " turn " + xa + "," + ya + "," + za + ","+ angle)) {
		System.out.println("ERROR: Could not turn " + itemName + "!");
		return false;
	}
	if (!act("attach leftHand " )) {
		System.out.println("ERROR: Could not dettach leftHand effector!");
		return false;
	}
	return true;
}
```

Χαρακτηριστικό παράδειγμα αυτής της λειτουργικότητας αποτελεί το παρακάτω παράδειγμα όπου εάν ο SARA agent κάνει sense το item door4a τότε ελέγχει την κατάσταση του και εάν αυτό είναι «κλειστό» τότε σταματάει το wandering, δρα πάνω του, πάει κοντά του και ενημερώνει τη βάση γνώσης του.

```
if (getKB().get("item", "door4a").size() != 0 ) {
	System.out.println("Find door4a");
	sense();
	if (state_ROTATED("door4a", 24.602118f, 0, -3.8830428f, 23.5f, 0, -3) ==
	STATE_OPEN_ROTATED) {
		System.out.println("door4a: Run STATE_OPEN_ROTATED)");
	}
	else if (state_ROTATED("door4a",24.602118f, 0, -3.8830428f, 23.5f, 0, -3) ==
	STATE_CLOSED_ROTATED) {
		if (wander==1){
			stopWandering();
			wander=2;
			System.out.println("door4a: Run STATE_CLOSED_ROTATED");
			ensureOpen_ROTATED("door4a", "Door4_a_access_p", "rotate",24.602118f,
			0, -3.8830428f, 23.5f,  0, -3f,0,1,0,-1.5f);
```

```
                    Vector<Vector<String>> Dlocations = getKB().get("location", "door4a");

                    // get the second argument of the first symbol instance found...
                    String Dlocation = Dlocations.get(0).get(1);

                    // the argument is of the form "x,y,z", parse it to get...
                    // individual coordinates...
                    String[] coords = Dlocation.split(",");
                    Dx = Float.valueOf(coords[0]);
                    Dy = Float.valueOf(coords[1]);
                    Dz = Float.valueOf(coords[2]);
                    sense();
                    update();
                    frame.repaint();
                    pathfind( 24,-2f);
                    startPathExecution();
                    System.out.println("PathExecuted");
                    sense();
                    update();
                    frame.repaint();
            }
        }
        else if (state_ROTATED("door4a", 224.602118f, 0, -3.8830428f,  23.5f, 0, -3) ==
        STATE_UNDEFINED_ROTATED) {
                System.out.println("door4a: Run STATE_UNDEFINED_ROTATED");
                sense();
                update();
                frame.repaint();
        }
        else {
                //do nothing
        }
}
```

### 7.3.3 Συναρτήσεις για κόμβους items τα οποία πραγματοποιούν κατά την κίνηση τους translation

Μία ακόμη λειτουργία την οποία καλείται να εκτελέσει ο SARA agent στα items είναι η μετατόπιση κόμβων των items όπως συνέβαινε και στην περίπτωση του housekeeper. Έτσι χρησιμοποιούνται οι αρχικοποιήσεις και οι συναρτήσεις isEqual(), state_nodes(), move_nodes(), ensureOpen_nodes(), ensureClosed_nodes() ακριβώς όπως στην περίπτωση του housekeeper και για τους ίδιους ακριβώς λόγους με μόνη διαφορά τον effector που χρησιμοποιείται στην move_nodes() καθώς ο SARA agent δεν διαθέτει hand effector. Έτσι έχει χρησιμοποιηθεί ο leftHand effector τυχαία ανάμεσα στους effector του SARA agent.

```
public boolean move_nodes(String itemName, String accessPointName, String functionName, float x, float y, float z) {
        if (!act("attach leftHand " + itemName + " " + accessPointName)) {
                System.out.println("ERROR: Could not attach leftHand effector to " +
                accessPointName + " accesspoint!");
                return false;
        }
        if (!act("select leftHand " +  " use " + functionName)) {
                System.out.println("ERROR: Could not select " + functionName + " function for move
                action!");
                return false;
        }
        if (!act("act leftHand "  + " use " + x + "," + y + "," + z )) {
                System.out.println("ERROR: Could not move " + itemName + "!");
                return false;
        }
        if (!act("attach leftHand " )) {
                System.out.println("ERROR: Could not dettach leftHand effector!");
                return false;
        }
        return true;
}
```

Χαρακτηριστικό παράδειγμα αυτής της λειτουργικότητας αποτελεί το παρακάτω παράδειγμα όπου εάν ο SARA agent κάνει sense το item chest_room4 τότε ελέγχει την κατάσταση του κόμβου που ορίζει το σύμβολο chestr4_in_up_symbol και εάν αυτό είναι «κλειστό» τότε σταματάει το wandering, το μετατοπίζει, πάει κοντά του και ενημερώνει τη βάση γνώσης του.

```
if (getKB().get("item", "chest_room4").size()!=0) {
        if (state_nodes("chestr4_in_up_symbol",  0,1, 0 , 0, 2.5f,0) == STATE_CLOSED) {
                if (wander==1){
                        stopWandering();
                        wander=2;
                        System.out.println("chestr4_in_up: Run State_CLOSED");
                        ensureOpen_nodes("chest_room4", "chestr4_in_up_symbol",
                        "chest_r4_revealmap_access_p", "translate",0, 1,0, 0, 2.5f, 0);
                        Vector<Vector<String>> Dlocations = getKB().get("location",
                        "chest_room4");
                        // get the second argument of the first symbol instance found...
                        String Dlocation = Dlocations.get(0).get(1);
                        // the argument is of the form "x,y,z", parse it to get...
                        // individual coordinates...
                        String[] coords = Dlocation.split(",");
                        Dx = Float.valueOf(coords[0]);
                        Dy = Float.valueOf(coords[1]);
```

```
                        Dz = Float.valueOf(coords[2]);
                        sense();
                        update();
                        frame.repaint();
                        pathfind(6,-9);
                        startPathExecution();
                        sense();
                        update();
                        frame.repaint();
                }
        } else if (state_nodes("chestr4_in_up_symbol",  0, 1, 0 , 0, 2.5f,0) == STATE_OPEN) {
                System.out.println("chestr4_in_up: Run State_OPEN");
                sense();
                update();
                frame.repaint();
        } else if (state_nodes("chestr4_in_up_symbol",   0, 1, 0 , 0, 2.5f,0) == STATE_UNDEFINED) {
                System.out.println("chestr4_in_up: Run State_OPEN");
                sense();
                update();
                frame.repaint();
        }else{
        }
}
```

### 7.3.4 Συναρτήσεις για κόμβους items τα οποία πραγματοποιούν κατά την κίνηση τους rotation

Επόμενη λειτουργία την οποία καλείται να εκτελέσει ο SARA agent στα items είναι η περιστροφή κόμβων των items όπως συνέβαινε και στην περίπτωση του housekeeper. Έτσι χρησιμοποιούνται οι αρχικοποιήσεις και οι συναρτήσεις isEqual(), state_nodes_rotated(), turn_nodes(), ensureOpen_nodes_rotated(), ensureClosed_nodes_rotated() ακριβώς όπως στην περίπτωση του housekeeper και για τους ίδιους ακριβώς λόγους με μόνη διαφορά τον effector που χρησιμοποιείται στην move_nodes() καθώς ο SARA agent δεν διαθέτει hand effector. Έτσι έχει χρησιμοποιηθεί ο leftHand effector τυχαία ανάμεσα στους effector του SARA agent.

```
public boolean turn_nodes (String itemName, String accessPointName, String functionName, float xa,
float ya, float za, float angle) {
        if (!act("attach leftHand " + itemName + " " + accessPointName)) {
                System.out.println("ERROR: Could not attach leftHand effector to " +
                accessPointName + " accesspoint!");
                return false;
        }
        if (!act("select leftHand " +  " use " + functionName)) {
                System.out.println("ERROR: Could not select " + functionName + " function for turn
                action!");
                return false;
        }
```

```
        if (!act("act leftHand "  + " use " + xa + "," + ya + "," + za + ","+ angle)) {
                System.out.println("ERROR: Could not turn " + itemName + "!");
                return false;
        }
        if (!act("attach leftHand " )) {
                System.out.println("ERROR: Could not dettach leftHand effector!");
                return false;
        }
        return true;
}
```

Χαρακτηριστικό παράδειγμα αυτής της λειτουργικότητας αποτελεί το παρακάτω παράδειγμα όπου εάν ο SARA agent κάνει sense το item table5 τότε ελέγχει την κατάσταση του κόμβου που ορίζει το σύμβολο table5_symbol_rotation και εάν αυτό είναι «κλειστό» τότε σταματάει το wandering, το περιστρέφει, πάει κοντά του και ενημερώνει τη βάση γνώσης του.

```
if (getKB().get("item", "table5").size()!=0) {
        if (state_nodes_rotated("table5_symbol_rotation",  1, 0, 0 , 0, 0,1) ==
        STATE_CLOSED_ROTATED) {
                if (wander==1){
                        stopWandering();
                        wander=2;
                        System.out.println("t17a: Run State_CLOSED");
                        ensureOpen_nodes_rotated("table5", "table5_symbol_rotation",
                        "table5_access_p",
                        "rotate",1, 0, 0 , 0, 0,1,1,0,0,1);
                        Vector<Vector<String>> Dlocations = getKB().get("location", "table5");
                        // get the second argument of the first symbol instance found...
                        String Dlocation = Dlocations.get(0).get(1);
                        // the argument is of the form "x,y,z", parse it to get...
                        // individual coordinates...
                        String[] coords = Dlocation.split(",");
                        Dx = Float.valueOf(coords[0]);
                        Dy = Float.valueOf(coords[1]);
                        Dz = Float.valueOf(coords[2]);
                        sense();
                        update();
                        frame.repaint();
                        pathfind(18, -9);
                        startPathExecution() ;
                        System.out.println("PathExecuted");
                        sense();
                        update();
                        frame.repaint();
                }
```

```
        }
        else if (state_nodes_rotated("table5_symbol_rotation",  1, 0, 0 , 0, 0,1)
        ==STATE_OPEN_ROTATED){
                System.out.println("t17a: Run State_OPEN");
                sense();
                update();
                frame.repaint();
        }
        else if (state_nodes_rotated("table5_symbol_rotation",   1, 0, 0 , 0, 0,1)
        ==STATE_UNDEFINED_ROTATED) {
                System.out.println("t17a: Run STATE_UNDEFINED");
                sense();
                update();
                frame.repaint();
        }else{
                //do nothing
        }
}
```

### 7.3.5 Συναρτήσεις για την σύγκριση συμβόλων

Τέλος μια πρόσθετη λειτουργικότητα την οποία καλείται να εκτελέσει ο SARA agent είναι αυτή της σύγκρισης συμβόλων. Ουσιαστικά αυτό που κάνει αυτή η συνάρτηση είναι να δέχεται ως ορίσματα δύο items και τα σύμβολα τους τα οποία θέλουμε να συγκρίνουμε, λαμβλανει τις τιμές από τα δύο σύμβολα και τις συγκρίνει μεταξύ τους.

```
public static int SYMBOL_STATE_EQUAL = 1;
public static int SYMBOL_STATE_NOTEQUAL = -1;

public int symbol_state(String itemName, String symbolName, String itemName2, String symbolName2)
{
        // get the location symbol instances for the specified item...
        Vector<Vector<String>> symbol_values = getKB().get(symbolName);

        // get the second argument of the first symbol instance found...
        String symbol_value = symbol_values.get(0).get(1);

        // the argument is of the form "x,y,z", parse it to get...
        // individual coordinates...
        String[] values = symbol_value.split(",");
        float x_value = Float.valueOf(values[0]);
        float y_value = Float.valueOf(values[1]);
        float z_value = Float.valueOf(values[2]);

        Vector<Vector<String>> symbol_values2 = getKB().get(symbolName2);

        // get the second argument of the first symbol instance found...
```

```
        String symbol_value2 = symbol_values2.get(0).get(1);

        // the argument is of the form "x,y,z", parse it to get...
        // individual coordinates...
        String[] values2 = symbol_value2.split(",");
        float x_value2 = Float.valueOf(values2[0]);
        float y_value2 = Float.valueOf(values2[1]);
        float z_value2 = Float.valueOf(values2[2]);
        if (x_value==x_value2 && y_value==y_value2 && z_value==z_value2) {
                return SYMBOL_STATE_EQUAL;
        }else{
                return SYMBOL_STATE_NOTEQUAL;
        }
}
```

Χαρακτηριστικό παράδειγμα αυτής της λειτουργικότητας αποτελεί η σύγκριση των χρωμάτων των γεωμετρικών σχημάτων των μηχανισμών ανοίγματος των πορτών με τα χρώματα των γεωμετρικών σχημάτων που βρέθηκαν στους χάρτες που ήταν κρυμμένοι στα μπαούλα.

```
//compare yellow cross button
        if (symbol_state("chest_room4", "rectangle_color_symbol","rect_mechanism",
        "yellowrect_color_symbol") == SYMBOL_STATE_EQUAL) {
                System.out.println("Yellow rect button...correct!!!");
                ensureOpen_nodes("rect_mechanism", yellowrect_symbol",
                "rectangle_mechanism_yellow_access_p", "translate", 0,0,1, 0,0,0);
                sense();
                update();
                frame.repaint();
        }else if (symbol_state("chest_room4", "rectangle_color_symbol","rect_mechanism",
        "yellowrect_color_symbol" )== SYMBOL_STATE_NOTEQUAL){
                System.out.println("Yellow rect button...is not correct!!!");
                ensureClosed_nodes("rect_mechanism", "yellowrect_symbol",
                "rectangle_mechanism_yellow_access_p","translate", 0,0,0, 0,0,-1);
                sense();
                update();
                frame.repaint();
        }
}
```

# 8. Συμπεράσματα – Περίληψη

Στην παρούσα μεταπτυχιακή διατριβή αναπτύχθηκε ένα ευφυές εικονικό περιβάλλον βασισμένο στην αναπαράσταση REVE και στις τεχνολογίες ανάπτυξης της. Το ίδιο περιβάλλον είχε επιχειρηθεί να αναπτυχθεί πιο πριν με χρήση των γλωσσών VRML και java. Με τη χρήση της γλώσσα VERL, της πλατφόρμας REVE Worlds και πρακτόρων ανεπτυγμένων σε java με βάση την REVE αναπαράσταση δημιουργήθηκε ένα πλήρες και λειτουργικό περιβάλλον με ευφυείς πράκτορες με σαφή λειτουργικότητα το οποίο πλεονεκτεί έναντι της προηγούμενης υλοποίησης λόγω της σαφώς μικρότερης δυσκολίας υλοποίησης, την ευελιξία και την επεκτασιμότητα της.

Αρχικά δημιουργήθηκε ο κόσμος / περιβάλλον σε VRML και στη συνέχεια μεταφέρθηκε σε VERL, μια τύπου xml γλώσσα για την απεικόνιση κόσμων με βάση την αναπαράσταση REVE. Κατά την μεταφορά αυτή έπρεπε να γίνουν κάποιες αλλαγές στα .wrl αρχεία ώστε να υποστηριχθεί η λειτουργικότητα της REVE αναπαράστασης. Η πλατφόρμα REVE Worlds υπολογίζει αυτόματα τα bounding boxes των items του κόσμου και με βάση αυτά ένας πράκτορας μπορεί να κινηθεί στον κόσμο. Βάση όμως X3D/VRML specification δεν είναι δυνατή η εξαγωγή πληροφορίας για γεωμετρικά αντικείμενα τύπου sphere, cylinder, box και cone από την στιγμή όπου έχει ξεκινήσει το rendering της σκηνής οπότε είναι αδύνατο να υπολογιστεί το bounding box. Για το λόγο αυτό μετατράπηκαν όλα αυτά τα geometry primitives σε IndexFacedSets ενώ ταυτόχρονα αφαιρέθηκαν και όλοι οι κόμβοι Collision καθώς δεν προσέφεραν πληροφορία αφού χρησιμοποιούνται για αλληλεπίδραση με τον χρήστη και όχι με κάποιον πράκτορα. Η μετατροπή σε IndexFacedSets έγινε με χρήση του προγράμματος VRMLPad το οποίο επιτρέπει την μετατροπή για αντικείμενα τύπου box, cylinder και cone, ενώ για την μετατροπή αντικειμένων τύπου sphere χρησιμοποιήθηκε ένα plugin ανεπτυγμένο από τον κ. Γιώργο Αναστασσάκη. Ιδιαίτερη προσοχή κατά τον ορισμό αντικειμένων στην VRML πρέπει να δοθεί στην ορθή σύνταξη τους χωρίς περιττή πληροφορία για την αναπαράσταση REVE όπως είναι παραδείγματος χάριν κάποιο translation το οποίο θα οριστεί εκ νέου στην γλώσσα VERL καθώς μπορεί να επηρεάσει τον υπολογισμό των bounding boxes. Τονίζεται ότι αυτό δεν αποτελεί λάθος της πλατφόρμας καθώς υπολογίζει ακριβώς αυτό που της ορίσαμε να υπολογίσει. Φυσικό επακόλουθο της μετατροπής των geometry primitives σε IndexFacedSets είναι η παραμόρφωση των textures τους από το tessellation στο οποίο υπόκεινται. Για την επίλυση του προβλήματος προτάθηκαν δύο τεχνικές, ανάλογα με το αρχικό geometry type του κάθε item, για geometries τύπου box χρησιμοποιήθηκε η τεχνική του textureCoordinate και για geometries τύπου cylinder, sphere και cone χρησιμοποιήθηκε η τεχνική του textureTransform. Τονίζεται ότι το πρόβλημα υπολογισμού των bounding boxes και τα προβλήματα που το ακολουθούν δεν αποτελεί bug της πλατφόρμας αλλά είναι απόρροια του X3D/VRML specification σχετικά με τον υπολογισμό του πεδίου size μετά το rendering.

Ένα πρόβλημα το οποίο παρατηρήθηκε είναι ότι ο ορισμός του κέντρου center ενός item στη VRML δεν περνάει στην πλατφόρμα και το σύστημα υπολογίζει το κέντρο του item εκ νέου. Αυτό μπορεί να δημιουργήσει πρόβλημα καθώς η εφαρμογή μιας συνάρτησης περιστροφής από πλευράς του πράκτορα σε ένα item δεν θα το περιστρέψει κατάλληλα με αποτέλεσμα είτε έλλειψη ρεαλισμού είτε πρόβλημα λειτουργικότητας. Για την επίλυση του προβλήματος στα items αυτά προστέθηκε στην VRML ένα νέος κόμβος ώστε να μετατοπίσει το κέντρο του item, προτείνεται όμως σε μια καινούρια έκδοση του REVE Worlds η προσθήκη δυνατότητας προσδιορισμού του κέντρου ενός item στο virtual model.

Όσων αφορά τις συναρτήσεις του access model AccessPointTranslateFunctionL1 και AccessPointRotateFunctionL1 με την X3DFieldFunctionL1 μια βασική διαφορά μεταξύ τους είναι ότι οι δύο πρώτες κατά την διάρκεια του animation μετατοπίζουν ή περιστρέφουν αντίστοιχα το item από την αρχική του θέση στην τελική σταδιακά, ενώ η τελευταία μετατοπίζει ή περιστρέφει τον κόμβο του item κατευθείαν στην τελική του θέση. Εάν αυτή τους η λειτουργικότητα δημιουργεί πρόβλημα ρεαλισμούς τον κόσμο προτείνεται η εφαρμογή πολλών διαδοχικών μετατοπίσεων ή περιστροφών από την αρχική στην τελική θέση έτσι ώστε να δημιουργείται το εφέ της σταδιακής κίνησης.

Τέλος όσων αφορά και πάλι το virtual model παρατηρήθηκε ότι ο ορισμός του scale ενός item ενώ οπτικά παρουσιάζεται στην πλατφόρμα με το σωστό μέγεθος, δεν περνάει η πληροφορία σε κάποιον πράκτορα με αποτέλεσμα να δημιουργεί μια αναληθή βάση γνώσης και να μην είναι σε θέση να λειτουργήσει αποτελεσματικά μέσα στον κόσμο. Για την επίλυση του προβλήματος ακολουθήθηκε μια συγκεκριμένη ακολουθία ορισμού των items στη VRML όσων αφορά των ορισμό των κόμβων scale η

οποία προσφέρει μια αποτελεσματική λύση του προβλήματος προτείνεται όμως η επίλυση του προβλήματος σε μια νέα έκδοση της πλατφόρμας.

Κατά την υλοποίηση του HouseKeeper του πράκτορα θεού δηλαδή του κόσμου αναπτύχθηκαν κάποιες συναρτήσεις για την υλοποίηση της λειτουργικότητας του πράκτορα για τις ακόλουθες ενέργειες:

- Συναρτήσεις για items τα οποία πραγματοποιούν κατά την κίνηση τους translation
- Συναρτήσεις για items τα οποία πραγματοποιούν κατά την κίνηση τους rotation
- Συναρτήσεις για κόμβους items τα οποία πραγματοποιούν κατά την κίνηση τους translation
- Συναρτήσεις για κόμβους items τα οποία πραγματοποιούν κατά την κίνηση τους rotation

Κατά τον ίδιο τρόπο αναπτύχθηκαν παρόμοιες συναρτήσεις για την υλοποίηση της συμπεριφοράς του SARA agent του πράκτορα παίχτη ουσιαστικά του κόσμου μας οι οποίες είναι οι ακόλουθες:

- Συναρτήσεις για items τα οποία πραγματοποιούν κατά την κίνηση τους translation
- Συναρτήσεις για items τα οποία πραγματοποιούν κατά την κίνηση τους rotation
- Συναρτήσεις για κόμβους items τα οποία πραγματοποιούν κατά την κίνηση τους translation
- Συναρτήσεις για κόμβους items τα οποία πραγματοποιούν κατά την κίνηση τους rotation
- Συναρτήσεις για την σύγκριση συμβόλων

Κατά την υλοποίηση του HouseKeeper χρειάστηκε να αντιμετωπιστούν κάποια προβλήματα για την ορθή λειτουργία του πράκτορα. Συγκεκριμένα ο housekeeper αντιδρά σε μια δράση του SARA agent αστραπιαία. Αυτό μπορεί να επιφέρει κάποια προβλήματα ανάλογα με την υλοποίηση καθώς μέχρι να ολοκληρώσει μια δράση ο SARA agent ο Housekeeper μπορεί να έχει εκτελέσει πολλαπλές επαναλαμβανόμενες αντιδράσεις στην πράξη του SARA agent. Το πρόβλημα εντοπίζονταν κυρίως σε μια υλοποίηση βασισμένη στον έλεγχο από πλευρά HouseKeeper της απόλυτης μετακίνησης ενός item και αντιμετωπίστηκε προτείνοντας μια υλοποίηση με βάση την σχετική μετακίνηση ενός item και τις αρχικές και τελικές του συντεταγμένες. Η ίδια λογική διατηρήθηκε και κατά την υλοποίηση του SARA agent καθώς εάν έλεγχε την απόλυτη μετακίνηση ενός item μέχρι την ολοκλήρωση της μετακίνησης ο πράκτορας θα δρούσε εκ νέου στο item.

Επιπρόσθετα παρατηρήθηκε μια ελάχιστη απόκλιση στον υπολογισμό της θέσης ενός item κατά την μετακίνηση τους συνεχώς αυξανόμενη από διαδοχικά animations η οποία μακροπρόθεσμα μπορεί να δημιουργήσει πρόβλημα λειτουργικότητας και για αυτό το λόγω ορίστηκα κάποιαν κατώφλια ελέγχου ώστε να μην επηρεάζονται οι πράξεις των πρακτόρων από αυτή τη διαφορά.

Επιπλέον πρέπει να τονίσουμε ότι οι συντεταγμένες του κάθε item διέπονται από το τοπικό σύστημα συνταγμένων του item και όχι από το σύστημα συντεταγμένων του κόσμου γεγονός που μπορεί να δυσχεραίνει την προσπάθεια κατά την υλοποίηση καθώς κάθε φορά θα πρέπει να ελέγχουμε το item στο οποίο θέλουμε δράσει ο πράκτορας. Προτείνεται σε μία επόμενη έκδοση του REVE Worlds τα δύο συστήματα συνταγμένων να ταυτίζονται.

Τέλος όσων αφορά ειδικά των SARA agent υλοποιήθηκε ένα απλό finite state machine ώστε να διαχειρίζεται τα threads που χρησιμοποιεί ο πράκτορας για να υλοποιήσει πράξεις μετακίνησης του. Παρατηρήθηκε πρόβλημα κατά την εναλλαγή των καταστάσεων του καθώς από τη στιγμή που ο πράκτορας έκανε stopWandering και τερματίζει το thread δεν ήταν δυνατή η εκ νέου εκκίνηση του με κλήση των συναρτήσεων StartPAthExecution, moveTo ή startWandering() καθώς σε αυτές οι λειτουργίες μπορούσαν να εκτελεστούν σε περίπτωση όπου το thread ήταν alive κάτι που δεν ίσχυε από την στιγμή που γίνονταν stopWandering(). Το πρόβλημα αυτό διορθώθηκε από τον κ. Γιώργο Αναστασσάκη με αλλαγή της βιβλιοθήκης της SARA.

Κατά την ενημέρωση της βάσης γνώσης του SARA agent και κλήση των συναρτήσεων update και frame.repaint παρατηρήθηκε ότι με την αύξηση των items στην kb ο πράκτορας αργεί να εκτελέσει τις παραπάνω συναρτήσεις. Για τον λόγο αυτό η κλήση των συναρτήσεων αυτών ελαχιστοποιήθηκε στο μικρότερο δυνατό ενώ ταυτόχρονα ανατήχθηκε μια δεύτερη έκδοση του εικονικού περιβάλλοντος το οποίο περιέχει μόνο τα δωμάτια πόρτες και διακόπτες. Προτείνεται σε μια νεότερη έκδοση της βιβλιοθήκης της SARA μια πιο αποδοτική ενημέρωση της βάσης γνώσης.

Τέλος ένα σημαντικό πρόβλημα στο οποίο ακόμη δεν έχει βρεθεί λύση αποτελεί ο εγκλωβισμός του SARA agent κάποιες φορές σε bounding boxes των items όταν ολοκληρώνει ένα segment πολύ κοντά σε αυτά.

Παρά τις όποιες δυσκολίες υλοποίησης και τα προβλήματα που παρουσιαστήκαν, τα οποία είτε ξεπεράστηκαν είτε αναμένουν λύση, η αναπαράσταση REVE, η πλατφόρμα REVE Worlds και η υλοποίηση πρακτόρων με βάση τις βιβλιοθήκες του Housekeeper και του SARA agent αποτελούν μια εξαιρετική λύση για την ανάπτυξη ευφυών εικονικών περιβαλλόντων τόσο από πλευράς ευκολίας υλοποίησης όσο και από πλευράς επεκτασιμότητας και ευελιξίας προσφέροντας ρεαλισμό και υποστήριξη διαδικασιών.

# 9. Βιβλιογραφία


- Αναπαράσταση και Λειτουργία Εικονικών Περιβαλλόντων, Διδακτορική Διατριβή, Γεώργιος Αναστασσάκης, Πειραιάς 2010
- Οδηγίες Εγκατάστασης και Δοκιμής REVE World Viewer, Γ. Αναστασσάκης, 29/03/2010
- REVE Worlds User Documentation, Virtual Model Definition, G. Anastassakis, 24-3-2010
- REVE Worlds User Documentation, Semantic Model Definition, G. Anastassakis, 24-3-2010
- REVE Worlds User Documentation, Access Model Definition, G. Anastassakis, 14-4-2010
- REVE Worlds User Documentation, Virtual Body Control Using the CLI, G. Anastassakis, 7-5-2010
- Η γλώσσα VRML, Νικήτας Σγουρός
- http://www.lighthouse3d.com
- http://mathforum.org