Θ.Παναγιωτόπουλος, Γ.Αναστασάκης, Γ.Αθανασόπουλος

Developing Applications in Second Life

Βοηθητικές Σημειώσεις για τα μαθήματα :

Κατανεμημένη Τεχνητή Νοημοσύνη

και

Εικονική Πραγματικότητα

Εισαγωγικό σημείωμα

Οι σημειώσεις που ακολουθούν προετοιμάστηκαν μετά από συνεργασία του διδάσκοντος με τους συνεργάτες του Εργαστηρίου, κ. Γ.Αναστασάκη και Γ.Αθανασόπουλο. Είναι ακόμη σε στάδιο ανάπτυξης και επεξεργασίας, μπορώ όμως να σας τις καταθέσω για να βοηθηθούν οι φοιτητές στην ανάπτυξη εφαρμογών στα πλαίσια εργασιών των μαθημάτων Κατανεμημένη Τεχνητή Νοημοσύνη και Εικονική Πραγματικότητα.

Εισαγωγή

Το 1999 ο Philip Rosedale σχημάτισε το Linden Lab. Έφτιαξε το Second Life το οποίο επιτρέπει στους χρήστες να βιώσουν μια πληθώρα εικονικών κόσμων. Στην πρώιμη μορφή του, η εταιρία δυσκολεύτηκε πολύ να κατασκευάσει μια εμπορική έκδοση του Hardware γνωστού και ως "The Rig" όπου σαν πρωτότυπο ήταν απλά μια ατσάλινη συσκευή με προσαρτημένα monitors. Η ιδέα αυτή εξελίχθηκε σε software με όνομα Linden World, στο οποίο οι χρήστες έπαιρναν μέρος σε παιχνίδια και μπορούσαν να αλληλεπιδράσουν τόσο με το χώρο γύρω τους όσο και με άλλους χρήστες η πράκτορες. Αυτή η προσπάθεια οδήγησε τελικά στο πλέον γνωστό Second Life.

Το Second Life είναι περισσότερα από ένα πράγματα. Πρόκειται για μία Open Source εφαρμογή που είναι κάτι μεταξύ εικονικού κόσμου , παιχνιδιού και εφαρμογής Chat. Αντίθετα με τα παραδοσιακά παιχνίδια υπολογιστών το Second Life δεν έχει συγκεκριμένο σκοπό ούτε κανόνες. Επίσης είναι περισσότερο από ένα απλό chat room καθώς παρέχει την δυνατότητα αλληλεπίδρασης με τον κόσμο. Επικεντρώνεται κυρίως στο να παρέχει ευκολία στους χρήστες να κατασκευάσουν αντικείμενα στον εκάστοτε κόσμο και να δώσουν την δικιά τους άποψη στον χώρο. Ο κόσμος δεν είναι προκατασκευασμένος κατά συνέπεια το πώς θα εξελιχθεί είναι κατά κύριο λόγο στα χέρια των χρηστών.(Για περισσότερες πληροφορίες σχετικά με το Second Life υπάρχει το official site http://secondlife.com/)

Με σκοπό να φτιάξουμε και εμείς εφαρμογές θα χρησιμοποιήσουμε τον server Open Simulator όπου θα μας επιτρέψει να αρχίσουμε να κατασκευάζουμε ανάλογα με τις ανάγκες μας. Όμως πριν από αυτό θα πρέπει να πούμε ορισμένα πράγματα για τον Open Simulator.

Ο Open Simulator είναι ένας open source server που μπορεί να υποστηρίξει εφαρμογές 3D πολλαπλών χρηστών που μπορεί να λειτουργήσει σε πολλά διαφορετικά περιβάλλοντα και σε πολλά διαφορετικής σύνθεσης Hardware συστήματα. Μπορεί να χρησιμοποιηθεί για την κατασκευή εικονικών κόσμων στους οποίους μπορούμε να έχουμε πρόσβαση μέσω διαφόρων clients, σε πολλαπλά πρωτόκολλα. Ο Open simulator επιτρέπει στον κατασκευαστή να χρησιμοποιήσει ότι τεχνολογίες τον εξυπηρετούν καλύτερα. Είναι

επεκτάσιμος καθώς είναι κατασκευασμένος σε γλώσσα C# η οποία τρέχει και σε περιβάλλον windows μέσω του .NET Framework αλλά και σε συστήματα Unix μέσω του Mono Framework.

Ο Open Simulator μπορεί να χρησιμοποιηθεί για να εξομοιωθούν περιβάλλοντα όπως το Second Life όπου ο χρήστης μπορεί να τα βιώσει μέσω οποιουδήποτε SL viewer (π.χ. Imprudence για τον οποίο θα μιλήσουμε αργότερα.) Ωστόσο δεν υποστηρίζει όλες τις λειτουργίες του Second Life.

Περισσότερες και πιο εξειδικευμένες πληροφορίες σχετικά με τον Open Simulator υπάρχουν στην ιστοσελίδα του http://opensimulator.org.

1. Second Life

1.1Ο κόσμος του Second Life.

Για αρχή πρέπει να δώσουμε μια ιδέα για το τι είναι ο κόσμος του Second Life και τι περιλαμβάνει. Πρόκειται για έναν τρισδιάστατο κόσμο στον οποίο ο χρήστης μπορεί να αγοράσει , να πουλήσει , να παίξει παιχνίδια τύχης , να αποκτήσει περιουσία, να φλερτάρει και πολλά άλλα πράγματα. Με λίγα λόγια μπορεί να έχει οποιουδήποτε τύπου συναλλαγή η συναναστροφή ή να κάνει οτιδήποτε μπορεί να επιθυμήσει σχεδόν, μόνος του ή ακόμη και με άλλους χρήστες. Σχεδόν οτιδήποτε μπορεί να κάνει στη πραγματική του ζωή είναι εφικτό στον κόσμο του Second Life αλλά ακόμη και πράγματα που δεν είναι εφικτά όπως η ζωή σε μια μεσαιωνική εποχή σε έναν φανταστικό κόσμο με μαγεία η ακόμη και η εξερεύνηση του διαστήματος όπως την έχει φανταστεί κάποιος.

Δεν πρόκειται για ένα απλό παιχνίδι καθώς ο σκοπός δεν είναι συγκεκριμένος όπως σε ένα παιχνίδι ενώ παρέχει στον χρήστη την δυνατότητα να επιδράσει στο περιβάλλον του και να το φέρει στα μέτρα του σε ορισμένες περιπτώσεις αυξάνοντας έτσι την αλληλεπίδραση ανθρώπου υπολογιστή. Μια εικόνα ενός κόσμου του Second Life φαίνεται στην παρακάτω φωτογραφία.

Το Second Life έχει ορισμένες επιρροές και κατά συνέπεια και ομοιότητες με τις σειρές παιχνιδιών “Sim City” και “The Sims” από τις Maxis και Electronic Arts. Στο Sim City οι παίκτες φτιάχνουν την δικιά τους πόλη ενώ στο The Sims ο εκάστοτε παίκτης ελέγχει έναν χαρακτήρα στις καθημερινές του δραστηριότητες. Εν μέρει το Second Life έχει μεγάλη αποδοχή στο κοινό λόγω αυτής της ομοιότητας του καθώς το The Sims είναι το πρώτο σε πωλήσεις στον κόσμο παιχνίδι εξαιτίας των χαρακτηριστικών του.

Επίσης παρουσιάζει ομοιότητες με Massively Multiplayer Online (MMO) παιχνίδια ρόλων όπως το πλέον γνωστό World of Warcraft παιχνίδι της Blizzard Entertainment. Οι χρήστες μπαίνουν στον κόσμο ελέγχοντας έναν Avatar με σκοπό να ολοκληρώσουν ορισμένες αποστολές μόνοι τους ή με άλλους χρήστες όπου θα τους παρέχουν experience points ώστε να ανεβάσουν τα επίπεδα δύναμης του χαρακτήρα που ελέγχουν. Παιχνίδια τέτοιου τύπου παρέχουν την δυνατότητα συνομιλίας μεταξύ των χρηστών. Αυτά όλα μπορούν να γίνουν και μέσω του Second Life

1.2 Αλληλεπίδραση χρήστη με τον κόσμο του Second Life.

Όπως αναφέρθηκε νωρίτερα στον κόσμο του Second Life δεν υπάρχουν κανόνες καθώς κάθε περιοχή μπορεί να έχει από διαφορετικούς φυσικούς νόμους έως και εντελώς διαφορετικές δυνατότητες που παρέχονται στους χρήστες.

Ο client από μόνος του παρέχει ορισμένες δυνατότητες στον χρήστη σε διάφορους τομείς οι οποίες μπορεί να αυξάνονται σε πλήθος ανάλογα τον viewer που χρησιμοποιούμε. Αρχικά ο κάθε χρήστης ξεκινάει με έναν εκ των δύο προεπιλεγμένων Avatars τους οποίους μπορεί είτε να αλλάξει επηρεάζοντας άμεσα την εμφάνιση τους μέσα από ορισμένα menu που παρέχονται από τον εκάστοτε viewer έως και να τον αντικαταστήσει πλήρως ακόμη και με κάτι όχι ανθρώπινο.(π.χ. μπορεί ένας χρήστης να χρησιμοποιήσει για avatar το μοντέλο κάποιου ζώου ή ακόμη και αντικειμένου.)

Ο default avatar δηλαδή άνθρωπος , έχει την δυνατότητα να κινηθεί περπατώντας η τρέχοντας στον χώρο καθώς επίσης και να πετάξει ελεύθερα. Πέρα από αυτά έχει την δυνατότητα να καθίσει είτε στο πάτωμα είτε σε κάποιο αντικείμενο πράγμα το οποίο πέρα από την εικαστική σημασία που έχει, έχει και την ιδιότητα να κάνει τον χαρακτήρα μας ως οντότητα να προσκολλάται στο αντικείμενο. Με αυτόν τον τρόπο έχουμε την δυνατότητα να εφαρμόσουμε ένα script σε ένα αντικείμενο και να επηρεάσει και τον χαρακτήρα που κάθεται πάνω. Χαρακτηριστικό παράδειγμα αποτελεί ένα ασανσέρ το οποίο πολλές φορές βοηθάει να το σχεδιάσουμε αναλόγως διότι ο avatar τείνει να πέφτει μέσα από τα πολύγωνα κατά την κίνηση.

Επίσης υπάρχει η δυνατότητα της συνομιλίας μέσω διαφόρων καναλιών (channels). Το chat channel χρησιμεύει σε περισσότερα πράγματα από απλή συνομιλία καθώς μπορεί να αποτελέσει τρόπο έναρξης κάποιου event π.χ. όπως θα δούμε αργότερα με το κατάλληλο script σε LSL μπορούμε να φτιάξουμε ένα αντικείμενο το οποίο αναμένει οδηγίες σε συγκεκριμένο κανάλι.

Η δυνατότητα επέμβασης στον κόσμο είναι πιο μεγάλη σε συγκεκριμένες περιοχές στις οποίες έχουν παραχωρηθεί στο χρήστη τα

δικαιώματα. Τυπικές τέτοιες περιοχές είναι αυτές που μπορεί να αγοράσει κανείς στο Second Life και που επιτρέπουν σε κάποιον να χτίσει σε αυτές ή το My Region ή όπως αλλιώς το ονομάσουμε όταν στήσουμε το περιβάλλον του Open Simulator. Σε αυτές τις περιοχές ο χρήστης έχει την δυνατότητα να χρησιμοποιήσει τα εργαλεία που παρέχονται από τον viewer και να φτιάξει από βασικά γεωμετρικά σχήματα (prims) ότι μπορεί να φανταστεί κανείς. Αυτά τα γεωμετρικά σχήματα μπορεί επιπλέον ένας χρήστης να τα εμπλουτίσει με μικρού συνήθως μήκους προγράμματα σε LSL και έτσι να προσθέσει κάποια ιδιότητα ή ακόμη και λειτουργία.

Για να ολοκληρώσουμε τις δυνατότητες που παρέχονται σε έναν avatar ή εικονικό πράκτορα πρέπει οπωσδήποτε να αναφέρουμε την δυνατότητα συναλλαγής μεταξύ των χρηστών του κόσμου ή μεταξύ χρήστη και εικονικού πράκτορα. Αρκετές εφαρμογές του εκάστοτε κόσμου στηρίζονται σε αυτή τη δυνατότητα καθώς κάποιος είναι ικανός ακόμη και να παρουσιάσει ή και να πουλήσει την δουλειά του μέσω ενός τέτοιου κόσμου. Κάθε χαρακτήρας είναι ικανός να κατέχει εικονικά περιουσιακά στοιχεία τα οποία πληρώνει συνήθως με SL dollars που αποτελούν το νόμισμα του Second Life ή ακόμη με τον κατάλληλο τρόπο μπορεί να οδηγήσει κάποιον σε συγκεκριμένη ιστοσελίδα και με πραγματικά χρήματα πλέον να ζητήσει η δικαιώματα χρήσης της δουλειάς του πωλητή ή ακόμη και εικονικές παροχές υπηρεσιών.

1.3 Δομή και οργάνωση του κόσμου

Ο κόσμος όντας μην συγκεκριμένος αλλά ορισμένος κυρίως από τον χρήστη μέσω τον όσων κατασκευάζει , δεν έχει μια δομή ωστόσο έχει μια συγκεκριμένη οργάνωση. Κάθε Server (Grid) αποτελείται από περιοχές (Regions) που έχουν μέγεθος 256 τετραγωνικά εικονικά μέτρα η κάθε μία. Οι συντεταγμένες των περιοχών αυτών στο σύνολο μπορούν να πάρουν τιμές από 0 έως (-1), δηλαδή πρόκειται για μία έκταση 281.475 τετραγωνικά χιλιόμετρα και περίπου 500 φορές το μέγεθος της γης. Κάθε περιοχή έχει έναν κάτοχο ή είναι κοινής χρήσης. Ο κάτοχος μιας περιοχής έχει την δυνατότητα να φτιάξει την περιοχή του όπως εκείνος

επιθυμεί ακολουθώντας κάποιους κανόνες καθώς επίσης και να ορίσει δικαιώματα επέμβασης από άλλους χρήστες.

Υπάρχει επίσης ένας ακόμη διαχωρισμός ο οποίος έχει να κάνει με την ηλικία του κατόχου του λογαριασμού καθώς αρκετές περιοχές είναι απαγορευμένες σε όλους κάτω των 18 χρονών.

1.4 Εφαρμογές στην εκπαίδευση

Πολλά πανεπιστήμια έχουν κατά καιρούς χρησιμοποιήσει το Second Life για εκπαιδευτικούς σκοπούς λόγω των δυνατοτήτων που παρέχει στους χρήστες. Λόγω της δυνατότητας χτισίματος στον κόσμο το πανεπιστήμιο του Περού St Martin de Porres σε δικιά του περιοχή σχεδιάζει και αναπαριστά αρχαιολογικής σημασίας κτίρια περουβιανού τύπου προς μελέτη από το 2008.

Επίσης για την οπτικοποίηση ορισμένων φαινομένων χρησιμοποιείται ο κόσμος του Second Life :

-Βιολογία :

Στο Genome Island μπορεί κάποιος να μάθει σχετικά με κύτταρα και χρωμοσώματα καθώς και διάφορες λειτουργίες τους.

-Χημεία :

Στο AAmerican Chemical Society's ACS Island μπορεί κάποιος να βιώσει οπτικά το πώς γίνονται οι χημικές αντιδράσεις. Έτσι παρέχεται στους σπουδαστές ένα ασφαλέστερο περιβάλλον για την μελέτη τέτοιων φαινομένων καθώς επίσης και η δυνατότητα να ‘’δουν’’ μια αξιόπιστη εξομοίωση.

- Φυσική/Ιατρική

Στο Virginia Tech's Slate οι φοιτητές μπορούν να μάθουν τόσο θεωρία όσο και να εξομοιώσουν ή και να διαβάσουν τα αποτελέσματα μιας αξονικής τομογραφίας.

Γενικότερα οτιδήποτε μπορεί να αναπαρασταθεί στο Second Life μπορεί να αποτελέσει υλικό για εκπαιδευτικούς σκοπούς. Έχοντας την δυνατότητα να δημιουργήσουμε οχήματα μπορούμε να φτιάξουμε εξομοιωτές πτήσης , οδήγησης αυτοκίνητου ή ακόμη πολύ απλά να κατασκευάσουμε puzzles.

1.5 Εφαρμογές σε άλλους τομείς.

- Διασκέδαση:

Το Second Life αποτελεί κατά κύριο λόγο τρόπο διασκέδασης για τον περισσότερο κόσμο καθώς επιτρέπει να «βιώσει» κανείς καταστάσεις ρεαλιστικά ενώ ταυτόχρονα παρέχει την δυνατότητα συνομιλίας. Σε αρκετές περιπτώσεις μπορεί να παρομοιαστεί με παιχνίδι για υπολογιστές στο οποίο ο εκάστοτε χρήστης παίζει μαζί με άλλα άτομα.

- Τέχνες:

Αρκετοί καλλιτέχνες βρίσκουν απασχόληση στο Second Life. Από εταιρίες που παρέχουν υλικό προς χρήση από τον κόσμο έως και απλούς χρήστες. Ένα πολύ ρεαλιστικά φτιαγμένο αντικείμενο ή μουσικό κομμάτι δεν αποτελεί λιγότερο τέχνη από ένα πραγματικό. Πέρα από αυτό η δυνατότητα δημιουργίας κόσμων ανοίγει νέους ορίζοντες στον τομέα των εικαστικών τεχνών.

- Marketing:

Η χρήση του Second Life από εταιρίες για διαφήμιση είναι αρκετά συνηθισμένη καθώς παρέχεται τόσο η δυνατότητα τοποθέτησης διαφημιστικών πινάκων όσο και η εικονική έκθεση του προϊόντος. Επίσης παρέχεται η δυνατότητα συναλλαγής με πραγματικά χρήματα γεγονός που διευκολύνει επιπλέον τις συναλλαγές.

- Τηλεπαρουσία:

Αποτελεί το κατάλληλο μέρος για να διεξαχθέν συνέδρια ή συναντήσεις μελών κάποιου οργανισμού οι οποίοι δεν είναι δυνατόν να παρευρίσκονται όλοι με φυσική υπόσταση σε συγκεκριμένο χώρο.

- Πρεσβείες:

Κάποιες πρεσβείες ανά τον κόσμο έχουν κατασκευάσει το δικό τους χώρο με σκοπό την εξυπηρέτηση και ενημέρωση του κόσμου για θέματα που αφορούν την εκάστοτε χώρα. Παραδείγματα τέτοιων πρεσβειών αποτελούν Μαλδίβες, Σουηδία, Σερβία, Σκόπια, Αλβανία Φιλιππίνες και άλλες.

2.Εγκαθιστώντας το περιβάλλον εργασίας

Η κατασκευή δικού μας εικονικού κόσμου απαιτεί εγκατάσταση Server. Έτσι θα εγκαταστήσουμε αρχικά τον OpenSimulator και τον Imprudence viewer όπου θα μας επιτρέψει να δούμε και να αλληλεπιδράσουμε με τον κόσμο μας. Αναλυτικότερη αναφορά στον Imprudence Viewer θα γίνει στο επόμενο κεφάλαιο.

2.1 Εγκαθιστώντας τον OpenSimulator σε Standalone Mode.

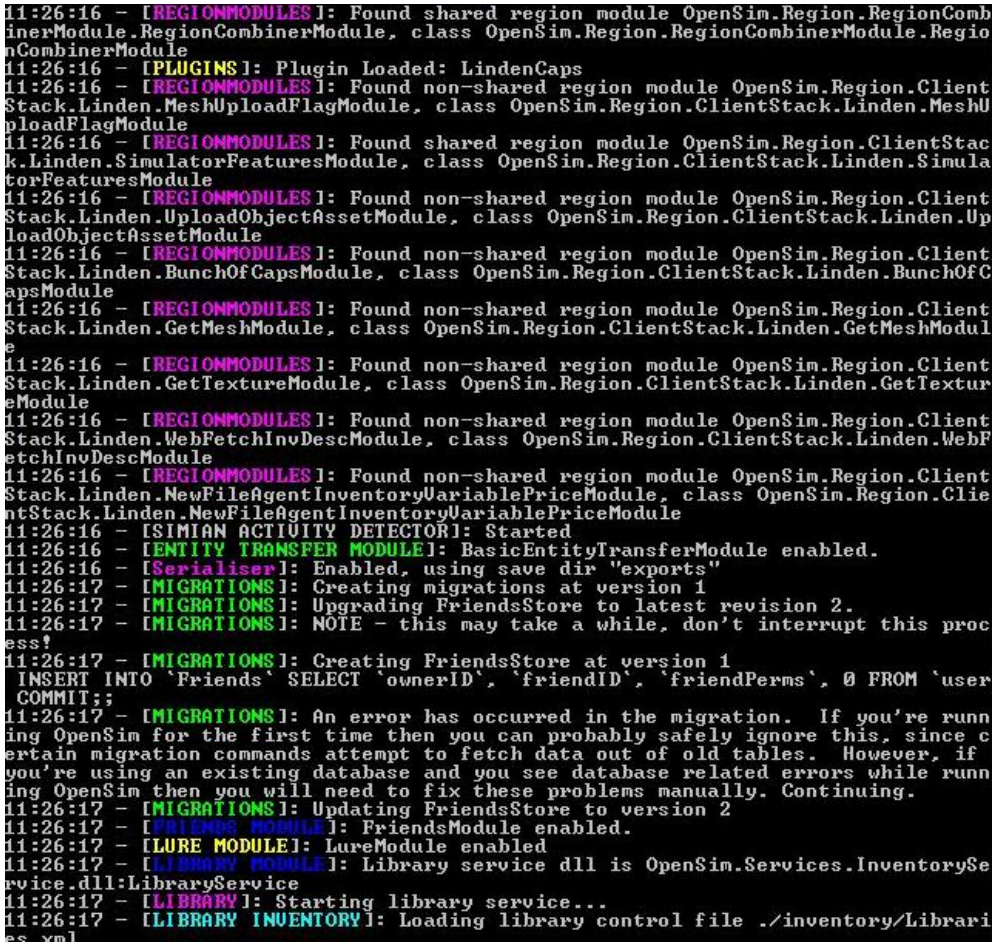
Για να εγκαταστήσουμε τον Open Simulator σε Standalone Mode πρέπει πρώτα να προμηθευτούμε το απαραίτητο λογισμικό του OpenSimulator και ενός Viewer ώστε να μπορέσουμε να συνδεθούμε και να δούμε τον κόσμο.

2.1.1 Εγκατάσταση και παραμετροποίηση του OpenSim

Το λογισμικό του OpenSimulator μπορεί να βρεθεί από την ιστοσελίδα του OSGrid http://www.osgrid.org/index.php/downloads ή από την ιστοσελίδα http://opensimulator.org/wiki/Download .

Μόλις ολοκληρωθεί η λήψη του OpenSim Server αποσυμπιέζουμε το αρχείο. Ο φάκελος bin περιέχει όλα τα αρχεία προς λειτουργία του. Τα βήματα για να παραμετροποιήσουμε τον Server είναι τα ακόλουθα:

- Εκκινούμε την εφαρμογή OpenSim.exe που βρίσκεται μέσα στο φάκελο bin και ένα παράθυρο DOS ανοίγει. Σε αυτό το παράθυρο αρχικά παρατηρούμε κάποια μηνύματα κατάστασης του server που εμφανίζονται. Μόλις σταματήσουν να εμφανίζονται νέα μηνύματα θα μας ζητηθούν ορισμένες πληροφορίες.(Για να αποδεχθούμε τις default ρυθμίσεις απλά πατάμε Enter.)



- Region Name[openSim Test]: Εδώ πληκτρολογούμε το όνομα που θέλουμε να έχει η περιοχή μας.

- Region UUID:

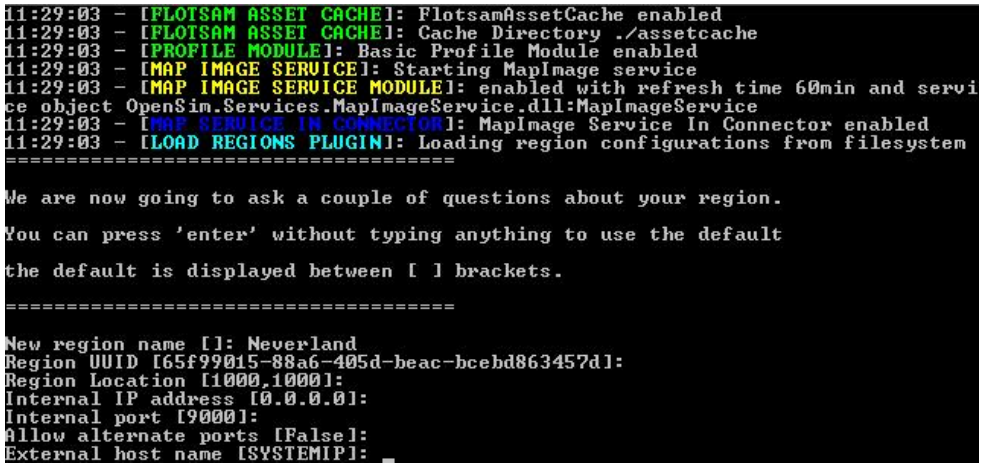
- Region Location[1000,1000]: Εδώ μας ζητείται να ορίσουμε την θέση της περιοχής μας στο Grid.Προτείνεται να το αφήσουμε ως έχει.

- Internal IP address[0.0.0.0]: Πρόκειται για την IP του Server που στην συγκεκριμένη περίπτωση θα είναι ο υπολογιστής μας οπότε το αφήνουμε στην προεπιλεγμένη τιμή.

- Internal Port[9000]: Εδώ ορίζουμε την θύρα επικοινωνίας.

- Allow alternate ports[false]: Επιλογή του αν θα επιτρέπουμε εναλλακτικές θύρες επικοινωνίας η όχι.

- External Host Name[SYSTEMIP]: Αυτή η επιλογή θα μας επιτρέψει αργότερα να ορίσουμε σύνδεση σε Grid Mode καθώς αυτό που μας ζητείται είναι η διεύθυνση προς σύνδεση. Συνήθως χρησιμοποιούμε κάποιο Domain Name.



- New Estate Name[My Estate]:

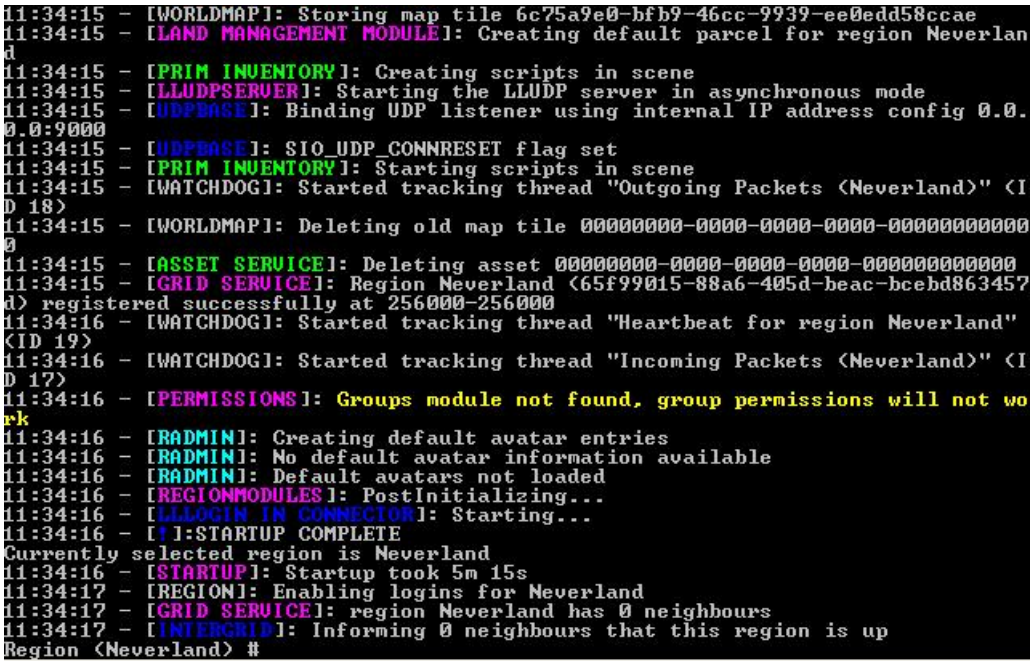
- Estate Owner first name[Test]: Σε αυτό το πεδίο μας ζητείται να πληκτρολογήσουμε το first name μέρος του ονόματος χρήστη που θα χρησιμοποιήσουμε στο λογαριασμό που έχει όλα τα δικαιώματα σε αυτήν την περιοχή.

- Estate Owner last name[User]: Ομοίως εδώ μας ζητείται να εισάγουμε το last name μέρος του ονόματος χρήστη.

- Password: και τον κωδικό πρόσβασης

- Email[]: Εισάγουμε το email μας αν θέλουμε.

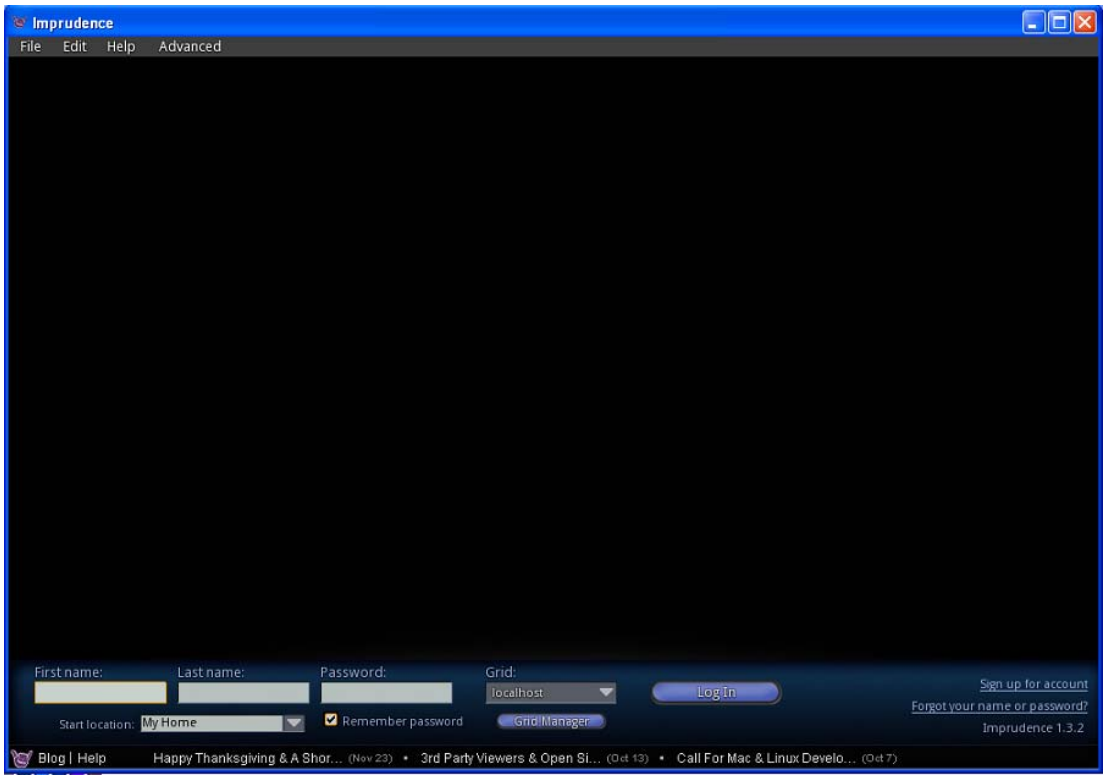
Μετά από αυτήν την διαδικασία ο server πρέπει να έχει παραμετροποιηθεί για χρήση στον προσωπικό μας υπολογιστή και θα πρέπει να καταλήξουμε σε μία εικόνα όπως η παρακάτω.



2.1.2 Εγκατάσταση του Imprudence Viewer

Για αυτήν την εργασία επιλέχθηκε προς χρήση ο Imprudence viewer. Για να προμηθευτούμε τον Imprudence viewer πηγαίνουμε στο http://wiki.kokuaviewer.org/wiki/Imprudence:Downloads και επιλέγουμε το αρχείο που αντιστοιχεί στην πλατφόρμα που χρησιμοποιούμε. Μόλις

ολοκληρωθεί η λήψη τρέχουμε το αρχείο του installer και ακολουθώντας τα βήματα ολοκληρώνουμε την εγκατάσταση.

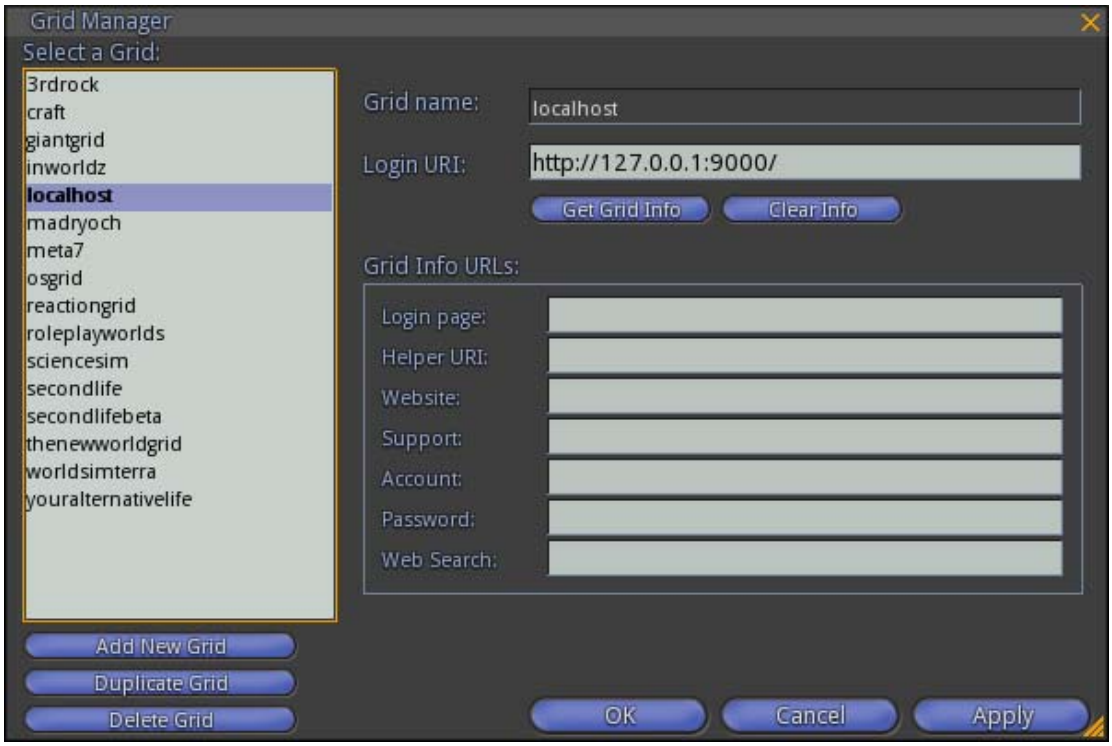


Εκκινούμε τον Open Simulator και μόλις ολοκληρωθούν τα status reports εκκινούμε τον Imprudence viewer. Εισάγουμε τα στοιχεία First Name,Last Name, Password και φροντίζουμε το Grid να έχει επιλεγμένο το localhost. Τέλος πατάμε Log In και εφόσον όλα έχουν ολοκληρωθεί σωστά πρέπει να καταφέρουμε να συνδεθούμε με έναν default avatar στην περιοχή που ορίσαμε ως My Region κατά την εγκατάσταση του OpenSimulator.

2.2 User-Oriented περιβάλλον.

Για την χρήση του Second Life απαιτείται μόνο η εγκατάσταση του Imprudence Viewer. Μόλις ολοκληρωθεί η εγκατάσταση μπορούμε να ορίσουμε σε πιο Grid θέλουμε να συνδεθούμε ,αλλάζοντας την επιλογή Grid μόλις εκκινήσουμε τον viewer, σε κάποια εκ των προεπιλεγμένων τιμών που αντιπροσωπεύουν ήδη ευρέως γνωστά Grids . Μπορούμε

επίσης να φτιάξουμε δικιά μας επιλογή για Grid το οποίο όμως πρέπει να υπάρχει για να καταφέρουμε να συνδεθούμε σε αυτό πατώντας στο Grid Manager.



Εδώ πατάμε Add New Grid και γράφουμε ένα όνομα στο Grid Name.Αμέσως μετά πηγαίνουμε στο Login URI: και γράφουμε το domain name που έχει ο Server στον οποίο θέλουμε να συνδεθούμε ή την IP address του ,ακολουθούμενα από : και μετά τον αριθμό της θύρας επικοινωνίας και πατάμε apply.

2.3 Εγκατάσταση πρόσθετων εργαλείων.

Αν επιθυμούμε να κάνουμε χρήση βιβλιοθηκών όπως η LibOpenMetaverse πρέπει να εγκαταστήσουμε έναν compiler για C# γλώσσα προγραμματισμού όπως παραδείγματος χάρη το Visual Studio 2010 ή ακόμη και το ελεύθερο για το κοινό Windows SDK από την διεύθυνση http://blogs.msdn.com/b/windowssdk/ και να κατεβάσουμε επίσης την βιβλιοθήκη που θα θέλαμε. Το LibOpenMetaverse μπορεί να βρεθεί εδώ : http://lib.openmetaverse.org/wiki/Download.

3. Imprudence Viewer

Ο Imprudence Viewer, πέρα από βελτιωμένα γραφικά σε σχέση με άλλους viewers, μας παρέχει επίσης και ορισμένες δυνατότητες σχετικά με τον κόσμο του Second Life , μερικές από τις οποίες θα εξεταστούν σε αυτό το κεφάλαιο. Οι δυνατότητες αυτές διαχωρίζονται στις ακόλουθες κατηγορίες:

3.1 Avatar

Μία από τις δυνατότητες που έχουμε είναι να αλλάξουμε την εμφάνιση του Avatar μας όπως εμείς επιθυμούμε. Για να αλλάξουμε τα χαρακτηριστικά της ενσάρκωσής μας πηγαίνουμε στο menu edit και επιλέγουμε appearance. Τότε εμφανίζεται ένα νέο παράθυρο με διάφορες καρτέλες και επιλέγουμε το αντίστοιχο τμήμα που θέλουμε να το αλλάξουμε και το αλλάζουμε καταλλήλως.

Μόλις ολοκληρώσουμε ένα τμήμα το αποθηκεύουμε κατά προτίμηση με διαφορετικό από το προεπιλεγμένο όνομα. Για να χρησιμοποιήσουμε αυτό το τμήμα , πατάμε στο πλήκτρο Inventory στο κάτω δεξιά μέρος του viewer και βρίσκουμε τον υποφάκελο που το έχει. Τότε πατάμε δεξί κλικ στο αντικείμενο και επιλέγουμε να το «φορέσουμε» (wear).

Στο menu Appearance μπορούμε επιπλέον να φτιάξουμε και αντικείμενα που ο χαρακτήρας μας θα φοράει ως ρούχα. Επιπλέον μπορούμε να κάνουμε εισαγωγή κάποιου avatar που έχουμε φτιάξει εμείς στο παρελθόν ή ακόμη και κάποιον που μπορεί να έχουμε αγοράσει πατώντας το πλήκτρο Import. Επίσης μας παρέχεται η δυνατότητα να αποθηκεύσουμε αντικείμενα προς μελλοντική χρήση με το πλήκτρο Export.

3.2 Πλοήγηση στο χώρο

Δύο από τους σημαντικότερους τρόπους αλληλεπίδρασης με το εικονικό μας περιβάλλον είναι η μετακίνηση μας στο περιβάλλον και η αλλαγή οπτικού πεδίου. Ο Imprudence viewer παρέχει τις παραπάνω δυνατότητες ως εξής:

- **Πλήκτρο up :** Μετακινεί την ενσάρκωσή μας προς τα εμπρός.
- **Πλήκτρο down:** Μετακινεί την ενσάρκωσή μας προς τα πίσω.

- **Πλήκτρο left:** Περιστρέφει την ενσάρκωσή μας προς τα αριστερά.

- **Πλήκτρο right:** Περιστρέφει την ενσάρκωση μας προς τα δεξιά.

- **Συνδυασμός πλήκτρων shift + left ή right:** Μετακινεί την ενσάρκωσή μας πλάγια αριστερά ή δεξιά αντίστοιχα.

Επίσης έχουμε την δυνατότητα να κάνουμε τον χαρακτήρα μας να πετάξει στο χώρο δίνοντας του την δυνατότητα να πετάξει προς τα επάνω ή προς τα κάτω και συνδυάζοντάς τα με τις παραπάνω λειτουργίες έχουμε ολοκληρωμένη δυνατότητα πλοήγησης στο εικονικό μας περιβάλλον.

- **Πλήκτρο E ή PgUp:** Μετακινεί την ενσάρκωσή μας προς τα πάνω αλλά πρέπει να κρατηθεί πατημένο για 2 δευτερόλεπτα ώστε να αρχίσει να αιωρείται όπου σταματήσουμε.

- **Πλήκτρο C ή PgDn:** Μετακινεί την ενσάρκωσή μας προς τα κάτω αλλά πρέπει να κρατηθεί πατημένο για 2 δευτερόλεπτα από την στιγμή που θα φτάσουμε στο πάτωμα για να επιστρέψουμε σε κατάσταση εδάφους.

- **Πλήκτρο F:** Αλλάζει την κατάσταση της κίνησης της ενσάρκωσής μας σε κατάσταση πτήσης ή εδάφους αντίστοιχα με το ποιά είναι ενεργή εκείνη την στιγμή. Όταν η κίνησή μας σταματάει σε κάποιο σημείο στον αέρα θα συνεχίσουμε να αιωρούμαστε. Αν πατήσουμε ξανά το F η κατάσταση αλλάζει πάλι σε κατάσταση εδάφους.

Τέλος έχουμε την δυνατότητα να περιστρέψουμε την κάμερα γύρω μας καθώς επίσης και να εστιάσουμε σε κάποιο συγκεκριμένο σημείο.

- **Μετακίνηση της ρόδας του ποντικιού προς τα πάνω:** Μεταφέρει την κάμερα πιο κοντά. Αν συνεχιστεί αυτό στο τέλος φτάνουμε σε μία κατάσταση στην οποία βλέπουμε από τα μάτια του avatar μας και μπορούμε να κοιτάξουμε τριγύρω μας χρησιμοποιώντας το ποντίκι έτσι ώστε όταν αλλάζει το οπτικό μας πεδίο να έχουμε περιστροφή και του

χαρακτήρα μας. Κατά συνέπεια μπορούμε να πλοηγηθούμε στον χώρο με τα πλήκτρα up και down και το ποντίκι μας.

- **Μετακίνηση της ρόδας του ποντικιού προς τα κάτω:** Μεταφέρει την κάμερα πιο μακριά από τα μάτια του avatar μας ώστε να έχουμε μια πιο πανοραμική θέα.

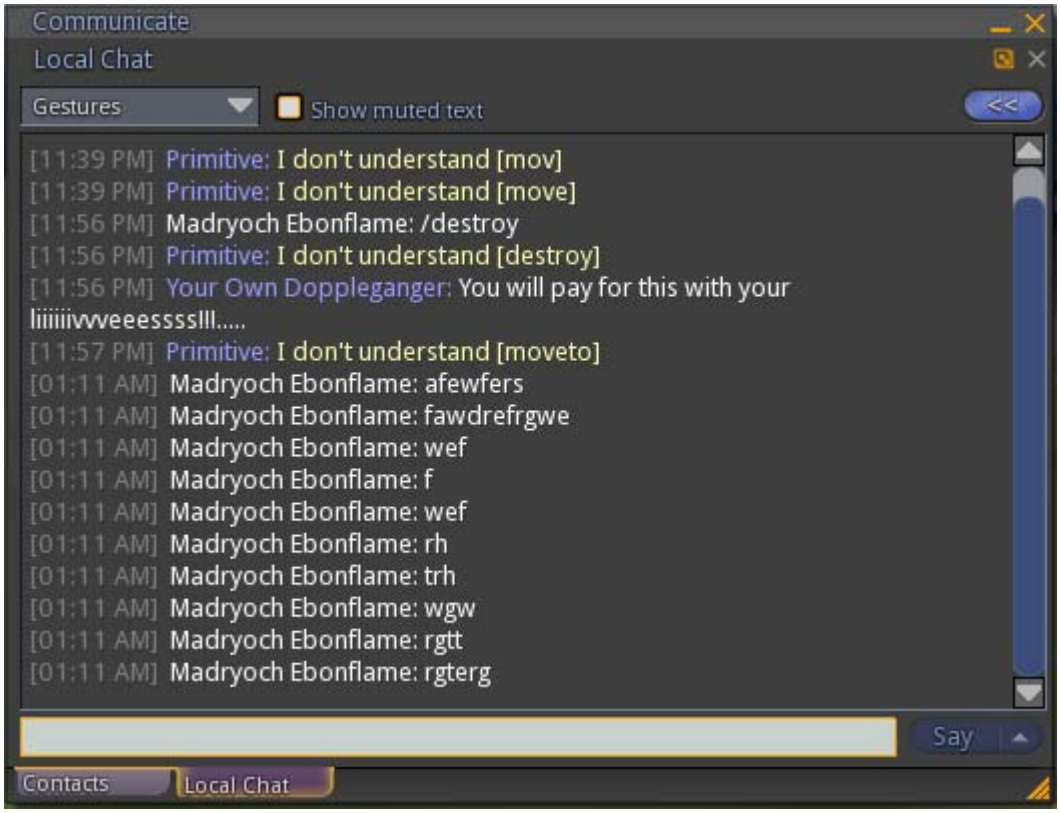
Η προεπιλεγμένη θέση της κάμερας βρίσκεται σε κατάσταση τρίτου προσώπου αρχικά. Επίσης σχετικά με την κάμερα υπάρχει και το μενού που μας επιτρέπει να μεταφέρουμε το οπτικό μας πεδίο με την χρήση του ποντικιού το οποίο βρίσκεται στο View→Camera Controls.

Τέλος , αν μετακινήσουμε τον δείκτη του ποντικιού σε κάποιο σημείο στο χώρο και κρατήσουμε πατημένο το ALT και το αριστερό πλήκτρο του ποντικιού , η κάμερα μετακινείται με σημείο αναφοράς αυτό το σημείο στο οποίο είναι πάνω ο δείκτης.

3.3 Δυνατότητα επικοινωνίας μέσω συνομιλίας.

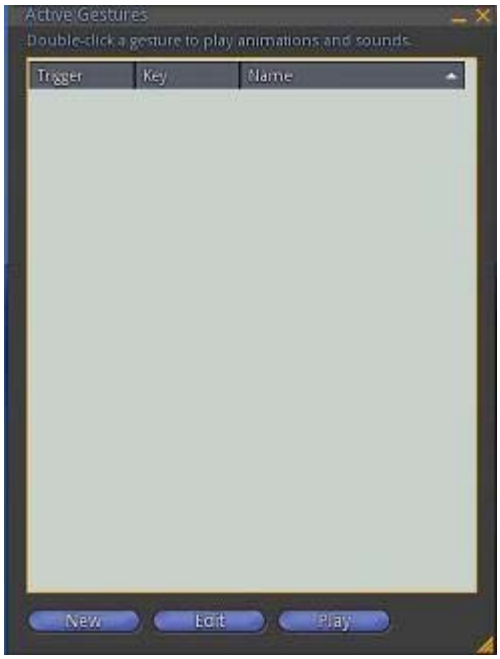
Άλλη μία σημαντική αλληλεπίδραση με τον κόσμο είναι η δυνατότητα συνομιλίας με άλλους χρήστες η και με αντικείμενα ή πράκτορες (Bots). Αυτή η συνομιλία γίνεται μέσω καναλιών. Πατώντας το πλήκτρο Enter και πληκτρολογώντας το μήνυμα μας κάνουμε το μήνυμα μας ορατό σε όλους τους χρήστες , πράκτορες και αντικείμενα που μπορεί να βρίσκονται στο χώρο εμβέλειας του εκάστοτε καναλιού που χρησιμοποιούμε. Για παράδειγμα το κανάλι say έχει περιορισμένη εμβέλεια και το βλέπουν μόνο όσοι χαρακτήρες βρίσκονται κοντά στην ενσάρκωση μας την στιγμή που το στέλνουμε. Για να μιλήσουμε σε κάποιο άλλο κανάλι απλά πατάμε **Enter** και γράφουμε / και έναν ακέραιο αριθμό που αντιπροσωπεύει το κανάλι.

Πατώντας το πλήκτρο Local Chat παρουσιάζεται ένα παράθυρο με το ιστορικό των συνομιλιών των χρηστών ή αντικειμένων που έχουν χρησιμοποιήσει ή χρησιμοποιούν το κανάλι say.



Επίσης πατώντας στο πλήκτρο Say μπορούμε να αλλάξουμε την εμβέλεια των όσων λέμε απλά επιλέγοντας Shout για να την αυξήσουμε ή whisper για να την μειώσουμε σε πολύ μικρή.

Τέλος , υπάρχει η δυνατότητα του να περιγράψουμε πιο παραστατικά ορισμένες ενέργειες που θα μπορούσαμε να κάνουμε σε αληθινές καταστάσεις γνωστές και ως χειρονομίες. Για να αποκτήσουμε πρόσβαση σε αυτές πρέπει να πατήσουμε στο πλήκτρο Gestures και να επιλέξουμε από εκεί κάποια χειρονομία από τις προϋπάρχουσες ή κάποια που έχουμε προσθέσει εμείς. Επιπλέον μπορούμε να φτιάξουμε μια νέα χειρονομία εμείς καθώς επίσης και να ορίσουμε κάθε φορά που μια συγκεκριμένη ακολουθία χαρακτήρων εμφανίζεται από εμάς , ο χαρακτήρας μας να εκτελεί αυτήν την χειρονομία.

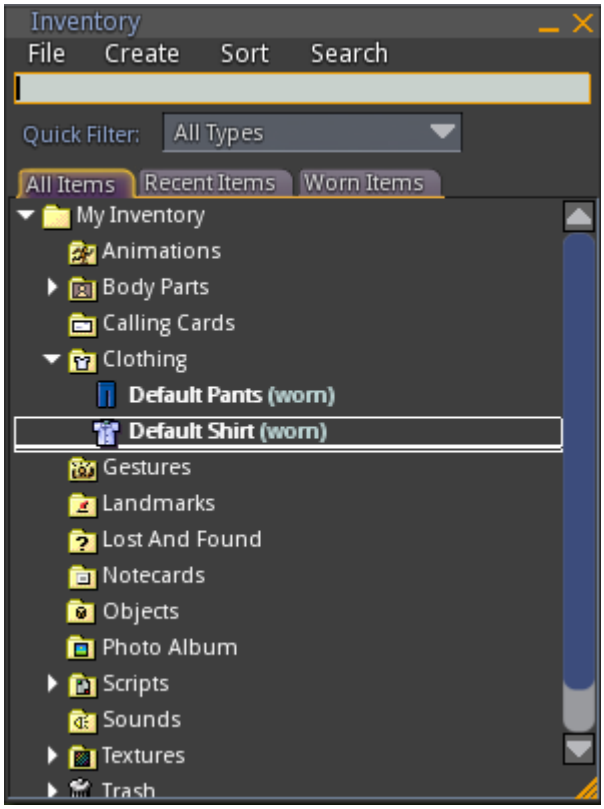


3.4 Χώρος αποθεμάτων (Inventory)

Όπως προαναφέρθηκε στο προηγούμενο κεφάλαιο, ένας χαρακτήρας μπορεί να κατέχει από αντικείμενα μέχρι και λεφτά. Με τον όρο αντικείμενα ,εννοούμε οτιδήποτε θα μπορούσε να μπει στον αποθεματικό του χώρο έως και πράγματα τα οποία έχει φτιάξει αυτός και έχει πλήρη δικαιώματα. Ακόμη και ένα αντικείμενο στο χώρο μπορεί να κατέχει αποθεματικό χώρο στον οποίο μπορούμε να ορίσουμε πράγματα που περιλαμβάνει. Εδώ θα παρουσιαστούν οι πέντε πιο σημαντικές κατηγορίες αντικειμένων που μπορεί κάποιος να έχει στον αποθεματικό του χώρο. Πατώντας στο κουμπί Inventory, κάτω δεξιά στην οθόνη του Imprudence ένα μενού εμφανίζεται και μας παρουσιάζει σε μορφή δέντρου του φακέλους και υποφακέλους του αποθεματικού χώρου που εξετάζουμε.

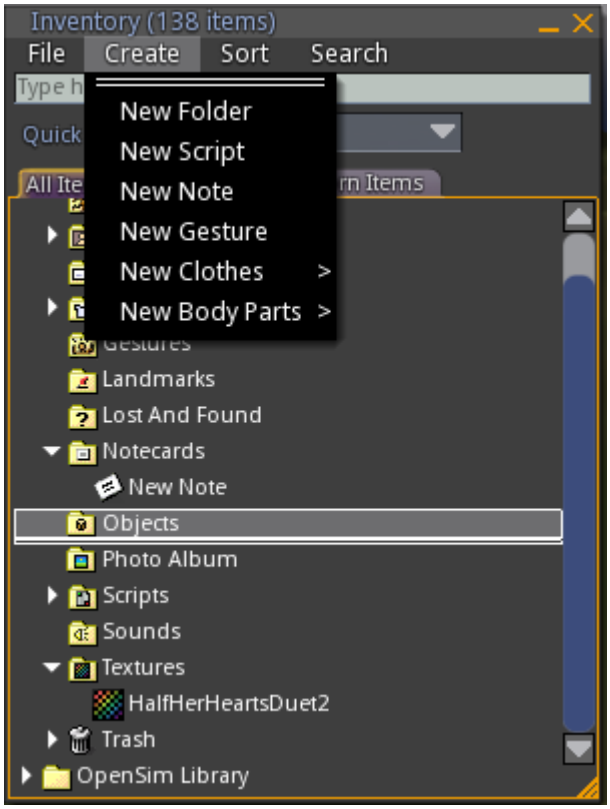
- **Κομμάτια σώματος.**

Εδώ μπορεί κανείς να φτιάξει τμήματα σώματος για την ενσάρκωσή του τα οποία μπορεί να αλλάξει μέσα από το μενού Appearance στο Edit. Πέρα από αυτό πατώντας δύο φορές το αριστερό πλήκτρο του ποντικιού μπορεί να φορέσει άμεσα το νέο αντικείμενο ή ακόμη πατώντας το δεξί κουμπί να επιλέξει να το βγάλει.

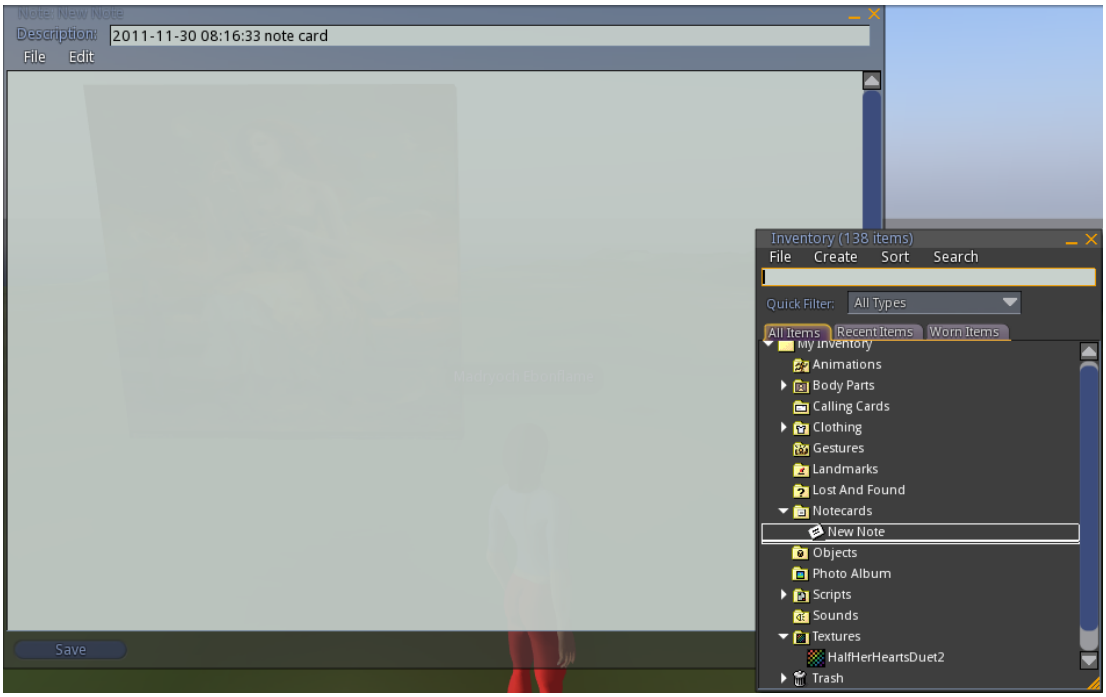


- **Σημειώσεις – Notecards**

Οι σημειώσεις μας βοηθούν να κρατήσουμε κάποιες πληροφορίες που εμείς επιθυμούμε για κάποιο αντικείμενο και μας παρέχουν όπως θα δούμε αργότερα , την δυνατότητα να επεξεργαστούμε ορισμένα ειδικά αντικείμενα. Η δημιουργία ενός Notecard γίνεται αν πατήσουμε το δεξί πλήκτρο το ποντικιού πάνω σε κάποιο σημείο στον αποθεματικό μας χώρο και επιλέξουμε New Note. Τότε θα εμφανιστεί ένα παράθυρο επεξεργασίας κειμένου στο οποίο μπορούμε να πληκτρολογήσουμε την σημείωση μας.

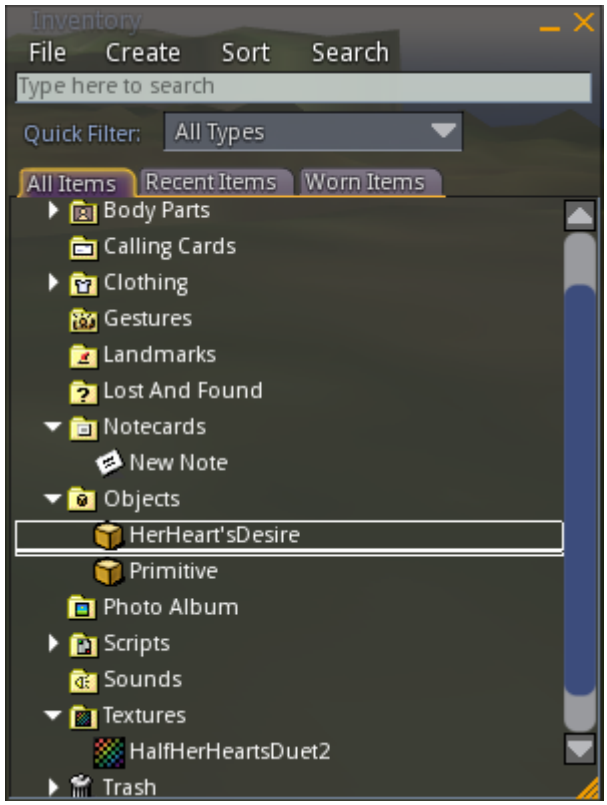


Πατώντας Save αποθηκεύουμε την σημείωσή μας, η οποία πλέον εμφανίζεται στον αποθεματικό μας χώρο με το όνομα New Note. Αν θέλουμε να μετονομάσουμε μία σημείωση απλά πατάμε το δεξί πλήκτρο του ποντικιού πάνω της και επιλέγουμε Rename.Πατώντας δυο φορές το αριστερό πλήκτρο του ποντικιού πάνω σε μία σημείωση το παράθυρο επεξεργασίας κειμένου εμφανίζεται και πάλι μαζί με τα όσα είχαμε κρατήσει σαν περιεχόμενά της.



- **Αντικείμενα (objects)**

Σε αυτό το τμήμα του αποθεματικού μας χώρου μπορούμε να τοποθετήσουμε αντικείμενα που παίρνουμε από τον χώρο ή αντικείμενα που έχουμε κατασκευάσει και προσθέσει στον αποθεματικό χώρο κάποιου φυσικού αντικειμένου. Πατώντας δεξί πλήκτρο του ποντικιού πάνω σε ένα αντικείμενο στο χώρο μας δίνεται η επιλογή Take. Με αυτήν την επιλογή παίρνουμε το αντικείμενο από τον χώρο και το αποθέτουμε στο Inventory μας. Μπορούμε να αλλάξουμε το όνομα ενός αντικειμένου με τον ίδιο τρόπο που αλλάζουμε το όνομα σε ένα Notecard.



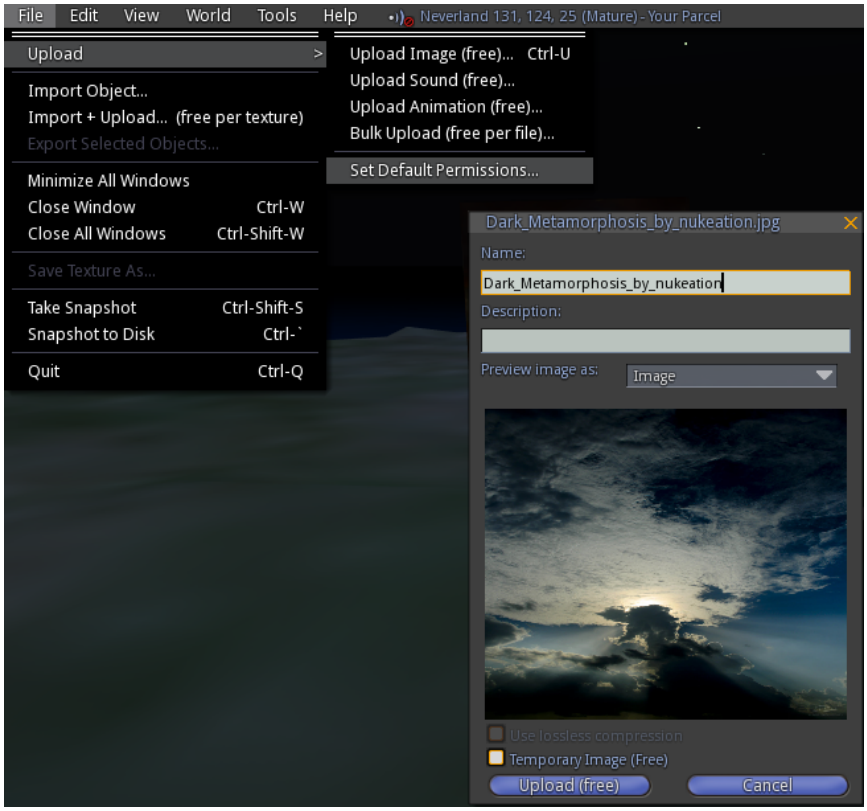
Τραβώντας το αντικείμενο από τον αποθεματικό μας χώρο στο περιβάλλον, φτιάχνεται ένα αντίγραφό του στον κόσμο αλλά το αντικείμενο παραμένει υπό την κατοχή μας στον αποθεματικό μας χώρο.

- **Τμήματα Κώδικα (Scripts)**

Μία σημαντική κατηγορία περιεχομένων του αποθεματικού χώρου είναι τα τμήματα κώδικα. Η χρησιμότητα τους είναι να προσθέτουν συμπεριφορές ή να αλλάζουν καταστάσεις σε αντικείμενα του χώρου. Αυτή η κατηγορία θα εξεταστεί αναλυτικότερα στο επόμενο κεφάλαιο στο οποίο θα έχουμε μια πιο εκτενή παρουσίαση των δυνατοτήτων της Linden Scripting Language (LSL) και της OpenSimulator Scripting Language (OSSL).

- **Εικόνες και υφές (Textures)**

Μια από τις δυνατότητες που μας παρέχονται , είναι να ανεβάσουμε κάποια εικόνα που έχουμε εμείς προς προσκόλληση σε επιφάνειες φυσικών αντικειμένων ώστε να δείχνουν πιο ρεαλιστικά. Επιλέγοντας File → Upload →Upload Image μπορούμε να επιλέξουμε μία εικόνα και να την περάσουμε σαν Texture στον αποθεματικό μας χώρο.



Τότε θα εμφανιστεί ένα παράθυρο που θα μας ζητάει να επιλέξουμε το όνομα που θέλουμε για το συγκεκριμένο Texture καθώς επίσης και μια περιγραφή. Πατώντας Upload αποθηκεύουμε την εικόνα μας στον αποθεματικό μας χώρο.

Πολλές κωδικοποιήσεις εικόνων (Filetype) μπορούν να γίνουν δεκτές προς αποθήκευση μεταξύ άλλων εικόνες jpeg και png. Αν επιθυμούμε οι εικόνες να έχουν διαφάνεια, πρέπει πρώτα να χρησιμοποιήσουμε το Photo Shop και να ορίσουμε τις περιοχές διαφάνειας που επιθυμούμε. Τέλος πρέπει να τις αποθηκεύσουμε σε μορφή PNG για να διατηρηθεί η διαφάνειά τους.

3.5 Χτίζοντας στον κόσμο του Second Life

Μια από τις καινοτομίες του Second Life σε σχέση με άλλες παλαιότερες εφαρμογές παρομοίου τύπου είναι η δυνατότητα που παρέχει στους χρήστες του να διαμορφώνουν τον εικονικό κόσμο. Η βασικότερη μέθοδος είναι ίσως το χτίσιμο στον κόσμο. Αλλά πριν ασχοληθούμε με το χτίσιμο πρέπει να δούμε πως μπορούμε να διαμορφώσουμε το έδαφος για να χτίσουμε πάνω σε αυτό μέσα από τον Imprudence viewer.

- **Διαμορφώνοντας το έδαφος**

Όπως και στην πραγματικότητα , έτσι και σε έναν εικονικό κόσμο για να μπορέσουμε να χτίσουμε πρέπει πρώτα να αρχίσουμε από τα θεμέλια του οικοδομήματός μας δηλαδή από την γη. Αυτό επιτυγχάνεται μέσω του Imprudence αν πατήσουμε το πλήκτρο B ή πάμε στο μενού View και επιλέξουμε Build. Μετά επιλέγουμε την πέμπτη καρτέλα  και εκεί βλέπουμε ορισμένες επιλογές με τις οποίες μπορούμε να επέμβουμε στο περιβάλλον. Αρχικά η περιοχή μας είναι ένα πολύ μικρό νησί.

Οι επιλογές που έχουμε είναι Select Land (επιλογή περιοχής), Flatten (εξίσωση), Raise (ανέγερση), Lower (βύθισμα), Smooth (Ομαλοποίηση), Roughen (εκτράχυνση), Revert(επαναφορά).Επίσης μπορούμε να επιλέξουμε την ένταση του φαινομένου που έχουμε και τον χώρο τον οποίο θα επηρεάζει.

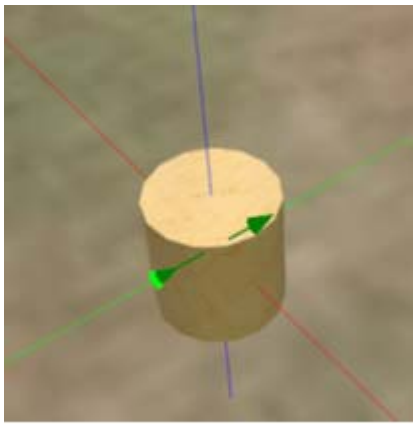
- **Βασικά Γεωμετρικά σχήματα**

Όπως σε κάθε τρισδιάστατη εφαρμογή , έτσι και στο Second Life τα πάντα στηρίζονται σε επιφάνειες και σε βασικά γεωμετρικά σχήματα. Έτσι για να φτιάξουμε το πάτωμα ενός κτιρίου απαιτείται πρώτα να κατασκευάσουμε διάφορα γεωμετρικά σχήματα τα οποία όταν τα ενώσουμε θα έχουμε το επιθυμητό αποτέλεσμα. Για να κατασκευάσουμε ένα βασικό γεωμετρικό σχήμα με τον Imprudence Viewer πατάμε B ή View→Build, επιλέγουμε την τέταρτη καρτέλα  και επιλέγουμε τον τύπο του σχήματος που θέλουμε να κατασκευάσουμε.

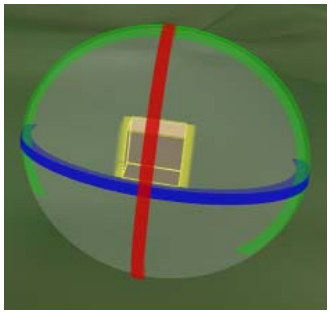
Αφού επιλέξουμε το σχήμα πατώντας το αριστερό πλήκτρο του ποντικιού δημιουργούμε ένα τέτοιο σχήμα στον χώρο μας με προεπιλεγμένες διαστάσεις και προσανατολισμό. Ορισμένες φορές είναι χρήσιμο να ορίζουμε ένα όνομα και μία περιγραφή της λειτουργίας που θέλουμε να εκτελεί το σχήμα στον χώρο. Επίσης αν θέλουμε να κατασκευάσουμε ένα αντίγραφο ενός υπάρχοντος σχήματος , αφού το επιλέξουμε , κρατώντας το πλήκτρο Shift και πατώντας πάνω σε ένα από τα βελάκια τον αξόνων που εμφανίζονται μπορούμε να «τραβήξουμε» ένα αντίγραφό του μακριά.

Επιλέγοντας την τρίτη καρτέλα  μπορούμε να επηρεάσουμε την θέση και τον προσανατολισμό ενός αντικειμένου.

Position (Θέση): Μας επιτρέπει να αλλάξουμε την θέση του αντικειμένου στο χώρο τραβώντας το στην θέση που θέλουμε. Απλά χρησιμοποιούμε τα βελάκια των αξόνων όταν έχουμε επιλεγμένο το αντικείμενο για να το τραβήξουμε προς την εκάστοτε κατεύθυνση.



Rotate (Περιστροφή): Είτε το επιλέγουμε από το μενού η απλά κρατάμε πατημένο πλήκτρο CTRL και περιστρέφουμε το αντικείμενο ως προς τον άξονα που εμείς επιθυμούμε.



Stretch (Επιμήκυνση) : Με αυτή την επιλογή επιτυγχάνεται η παραμόρφωση ενός αντικειμένου ως προς κάποιον άξονα ή ακόμη και συνολική παραμόρφωση σε όλες τις διαστάσεις(μεγέθυνση, σμίκρυνση). Η επίτευξη αυτού γίνεται αν επιλέξουμε αυτή την λειτουργία ή κρατήσουμε πατημένα το CTRL και το SHIFT και μετά επιλέξουμε να τραβήξουμε κάποιο από τα χρωματισμένα κυβάκια (παραμόρφωση ως προς έναν άξονα) ή κάποιο από τα γκρι (μεγέθυνση, σμίκρυνση).

Σημαντική επίσης είναι και δυνατότητα να ενοποιήσουμε κάποια αντικείμενα με το πλήκτρο Link. Για να το κάνουμε αυτό επιλέγουμε τα αντικείμενα που θέλουμε να ενοποιήσουμε κρατώντας πατημένο το Shift και μετά επιλέγουμε Link. Πλέον το τελευταίο αντικείμενο που προσθέσαμε αποτελεί την ρίζα του σχήματος και το ενοποιημένο αντικείμενο έχει το όνομα και την περιγραφή της ρίζας. Με την ενοποίηση ότι άλλη αλλαγή κάνουμε στο αντικείμενο εφαρμόζεται σε όλα τα μέλη του συνόλου. Για να επεξεργαστούμε κάποιο μόνο του επιλέγουμε Edit Linked Parts.

Λίγο πιο χαμηλά στην καρτέλα υπάρχουν και άλλες επιλογές. Μια από αυτές είναι η Object(αντικείμενο) στην οποία βλέπουμε ορισμένες επιλογές που μας παρέχονται σχετικά με το αντικείμενο.

Locked: Κλειδώνει το αντικείμενο και δεν επιτρέπει αλλαγές σε αυτό.

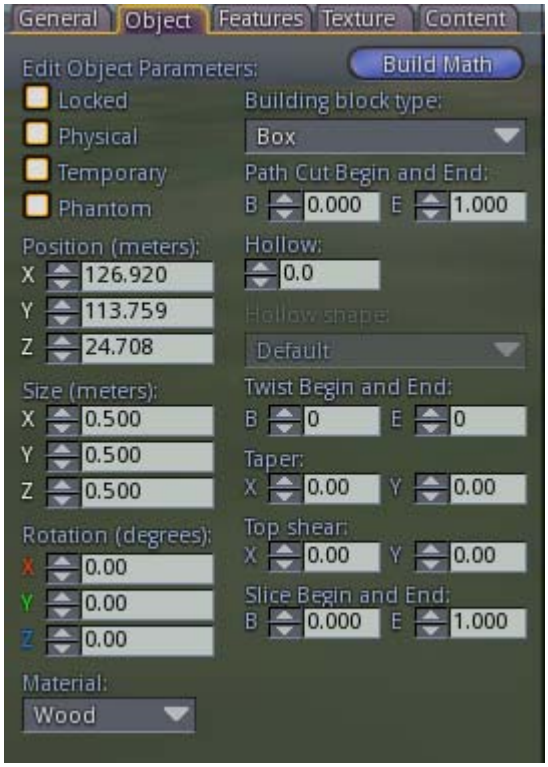
Physical: Θέτει το αντικείμενο υπό την επιρροή του Physics Engine του Second Life.

Temporary: Κάνει το αντικείμενο προσωρινό.

Phantom: Κάνει το αντικείμενο να μην έχει φυσική υπόσταση στον κόσμο κατά συνέπεια μπορούμε να περάσουμε από μέσα του. Είναι χρήσιμη σαν επιλογή αν θέλουμε να φτιάξουμε για παράδειγμα μια περιοχή που ο φωτισμός είναι πιο έντονος και συγκεντρωμένος.

Position: Εδώ μπορούμε να ορίσουμε επακριβώς την θέση του αντικειμένου στο χώρο βάζοντας συντεταγμένες.

Size: Ορίζουμε τις διαστάσεις του σχήματος. Σημαντική προσθήκη του Imprudence Viewer εδώ είναι η δυνατότητα που παρέχει να



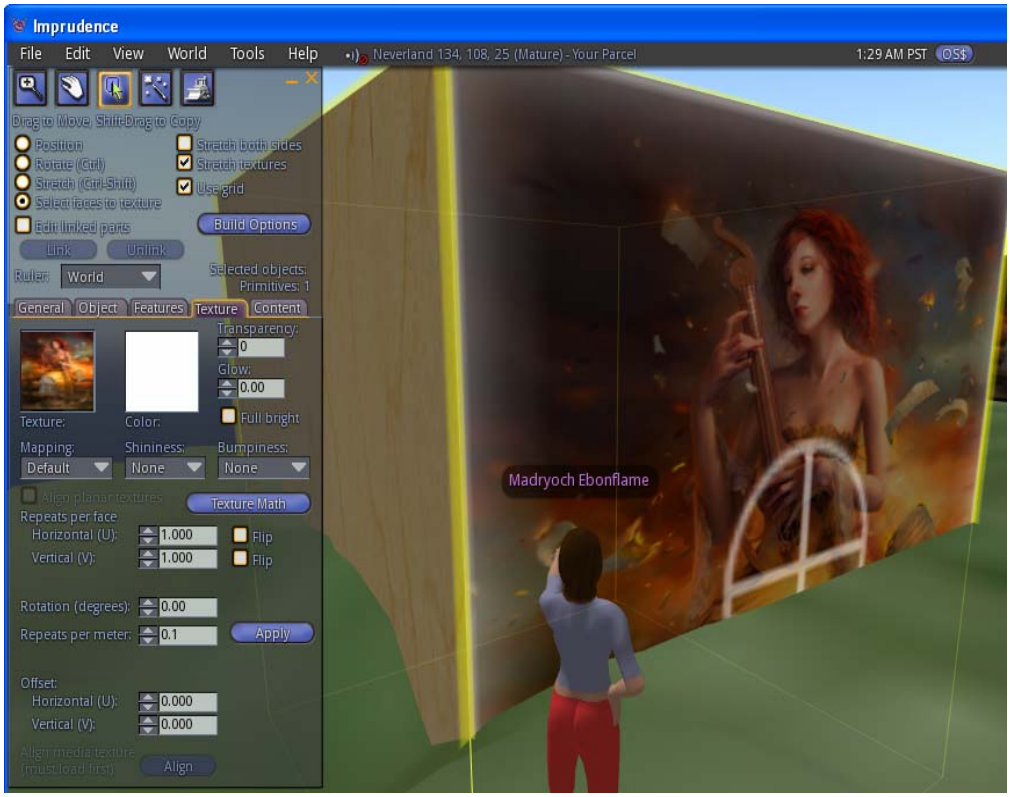
κατασκευάσουμε Mega Prims δηλαδή γεωμετρικά σχήματα που ο viewer του Second Life δεν επιτρέπει.

Αυτό βέβαια ισχύει μόνο αν ο Server που χρησιμοποιείται το επιτρέπει αυτό. Οι Server του Second Life δεν επιτρέπουν MegaPrims αλλά ο Open Simulator επιτρέπει.

Rotation: Επιτρέπει να ορίσουμε σε μοίρες περιστροφή ως προς τον εκάστοτε άξονα.

- **Προσθέτοντας υφές σε ένα αντικείμενο (Textures)**

Για περισσότερο ρεαλισμό μπορούμε να προσκολλήσουμε κάποια εικόνα σε μια ή περισσότερες επιφάνειες ενός γεωμετρικού σχήματος καθώς επίσης και να αλλάξουμε την υφή του. Πατώντας στην επιλογή Textures του viewer επιλέγουμε τόσο την εικόνα όσο και την διαμόρφωση της επιφάνειας που επιθυμούμε να αλλάξουμε καθώς επίσης και το χρώμα. Αν προσπαθήσουμε να επιλέξουμε μία εικόνα η ένα άλλο εφέ τότε αυτόματα εφαρμόζεται σε ολόκληρο το γεωμετρικό σχήμα. Αν θέλουμε να εμφανίζεται μόνο σε συγκεκριμένες επιφάνειες επιλέγουμε Select faces to texture και επιλέγουμε την επιφάνεια που θέλουμε.



Στην καρτέλα “Textures” βλέπουμε επίσης τις επιλογές Repeats per face horizontal και Vertical που ορίζει πόσες φόρες θα επαναλαμβάνεται το texture , rotation που ορίζει αν η υφή θα εφαρμοστεί αφού πρώτα υποστεί μια περιστροφή η εικόνα και τέλος το offset που ορίζει από που θα ξεκινάει η υφή πάνω στο αντικείμενο.

Επιπλέον υπάρχουν οι επιλογές Transparency που ορίζουμε ποσοστό διαφάνειας του αντικειμένου , Glow που ορίζει την φωτεινότητα του σχήματος σε ποσοστό , Shininess που ορίζει την γυαλάδα του και τέλος Bumpiness που ορίζει πόσο τραχύ θα είναι το αντικείμενο.

- **Περιεχόμενα**

Κάθε αντικείμενο μπορεί να περιέχει και άλλα αντικείμενα ή σημειώσεις που έχουμε στον αποθεματικό μας χώρο ή εικόνες ,animations και άλλα. Αν το αντικείμενο είναι προγραμματισμένο να έχει

και κάποια συμπεριφορά τότε είναι πιθανόν να περιέχει και κάποιο αντικείμενο script.

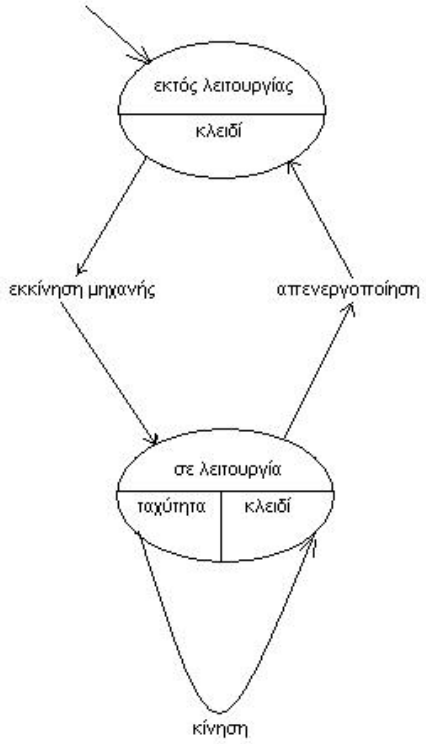
- **Δικαιώματα**

Όσα πράγματα χτίσουμε στον κόσμο είναι προσβάσιμα για οποιαδήποτε αλλαγή από εμάς. Δεν συμβαίνει το ίδιο όμως και για άλλους χρήστες. Για να τα κάνουμε προσβάσιμα από όλους πηγαίνουμε στην καρτέλα General και επιλέγουμε να τους παρέχουμε δικαιώματα.

4. Εισαγωγή στην LSL και OSSL.

Η LSL(**L**inden **S**cripting **L**anguage) όπως και η OSSL(**O**pen **S**imulator **S**cripting **L**anguage**)** είναι γλώσσες κατασκευασμένες κατά το πρότυπο των μηχανών καταστάσεων (Finite State Machines FSM ή Finite State Automaton) και είναι οι γλώσσες που χρησιμοποιούνται κατά κόρον στην ανάπτυξη υλικού και συμπεριφορών στον κόσμο του Second Life. Πρέπει όμως να δώσουμε μερικές πληροφορίες ακόμη για το τι είναι μια μηχανή καταστάσεων και το τι περιλαμβάνει ώστε να κατανοήσει ο εκάστοτε χρήστης πώς να αναπτύξει κώδικα επιτυχώς.

Μια μηχανή καταστάσεων είναι μια αφηρημένη μηχανή με πεπερασμένο αριθμό καταστάσεων. Κάθε φορά η μηχανή ξεκινά βρισκόμενη σε μια κατάσταση την οποία ονομάζουμε αρχική. Ανά πάσα χρονική στιγμή η μηχανή μπορεί να βρίσκεται σε μία και μόνο κατάσταση και αναμένει να δεχτεί ορισμένα δεδομένα εισόδου τα οποία είτε μεταφέρουν την μηχανή στην ίδια κατάσταση ή ακόμη και σε διαφορετική. Οι μηχανές καταστάσεων ως μοντέλο βοηθούν να περιγραφούν παραστατικά και να υλοποιηθούν πολλά θέματα που καλείται να αντιμετωπίσει η επιστήμη της πληροφορικής αλλά και άλλες επιστήμες. Για να κατανοήσουμε το μοντέλο, ας φανταστούμε την μηχανή ενός αυτοκινήτου. Η μηχανή για απλοποίηση θα θεωρήσουμε ότι έχει δύο καταστάσεις. Κατάσταση σε λειτουργία (ON) και κατάσταση εκτός λειτουργίας(OFF). Όταν γυρίζουμε το κλειδί η μέθοδος με την οποία η μηχανή παίρνει μπροστά ξεκινάει και η μηχανή μεταφέρεται στην κατάσταση ‘’σε λειτουργία(ON)’’. Αντίθετα όταν η μηχανή λειτουργεί αν γυρίσουμε το κλειδί ξανά σβήνει δηλαδή μεταφέρεται στην κατάσταση ‘’εκτός λειτουργίας(OFF)’’. Αν ενώ η μηχανή βρίσκεται σε λειτουργία, εμείς βάλουμε ταχύτητα και πατήσουμε γκάζι η μηχανή αρχίζει να κινείται. Αυτό θα μπορούσε να χαρακτηριστεί ως διαφορετικός τύπος δεδομένων εισόδου.



Εδώ φυσικά θα μπορούσε να είχε γίνει διαφορετικός διαχωρισμός καταστάσεων όπως για παράδειγμα από την κατάσταση λειτουργίας με την είσοδο ταχύτητα να πηγαίνει σε νέα κατάσταση, την κατάσταση κίνησης και εκεί με την είσοδο γκάζι να κινείται. Θα μπορούσαμε ακόμη να ορίσουμε και μία ξεχωριστή κατάσταση για την κάθε ταχύτητα.

4.1 Δομή LSL και OSSL

Η OSSL είναι μια επέκταση της γλώσσας LSL και τον δυνατοτήτων της. Κατά συνέπεια έχει την ίδια δομή όπως και η LSL.

4.1.1 Καταστάσεις

Οι καταστάσεις στην LSL παριστάνονται από τμήματα κώδικα τα οποία έχουν τον έλεγχο την δεδομένη χρονική στιγμή. Για να ορίσουμε μία κατάσταση αρκεί να γράψουμε το ακόλουθο τμήμα κώδικα:

```
state Όνομα_Κατάστασης
{
```

…

}

Κάθε πρόγραμμα(script) έχει υποχρεωτικά μια αρχική κατάσταση που ονομάζεται default. Αυτή η κατάσταση είναι η μόνη που έχει διαφορετική σύνταξη κατά τον ορισμό της. Για να ορίσουμε την κατάσταση default χρησιμοποιούμε την ακόλουθη σύνταξη.

default

{

…

}

Δεν υπάρχει περιορισμός στον αριθμό καταστάσεων που θέλουμε να έχει το πρόγραμμά μας με εξαίρεση τον περιορισμό του αποθηκευτικού χώρου. Για να μεταβούμε από μία κατάσταση σε μία άλλη αρκεί να γράψουμε την λέξη state ακολουθούμενη από το όνομα της κατάστασης που θέλουμε να μεταβούμε και προσθήκη στο τέλος του χαρακτήρα ; που δηλώνει στην LSL ότι αυτό είναι το τέλος της εντολής.

state Όνομα_Κατάστασης;

4.1.2 Συμβάντα (Events)

Τα συμβάντα (Events) θα μπορούσαμε να πούμε ότι είναι τα δεδομένα εισόδου σε κάθε κατάσταση στο πρόγραμμά μας. Στο Second Life υπάρχουν πολλά πιθανά συμβάντα που μπορούμε να ορίσουμε σε κάθε κατάσταση. Μερικά από αυτά θα τα μελετήσουμε αργότερα. Όλα είναι της μορφής αν συμβεί η συνθήκη που θέτουμε τότε πρέπει να εκτελεστεί κάποιο συγκεκριμένο τμήμα κώδικα.

4.1.3 Σχόλια

Για να προσθέσουμε σχόλια στον κώδικά μας πληκτρολογούμε δυο φορές την κάθετο / .Οτιδήποτε πληκτρολογηθεί σε αυτήν την σειρά δεξιότερα από τις δύο καθέτους , απλά αγνοείται κατά την εκτέλεση.

```
//Αυτό είναι ένα σχόλιο.
```

4.1.4 Μεταβλητές

Μια μεταβλητή αντιπροσωπεύει μία συγκεκριμένη θέση στη μνήμη του υπολογιστή. Σε αυτήν την θέση μπορούμε να αποθηκεύσουμε κάποια τιμή την οποία μετά έχουμε την δυνατότητα να ανακτήσουμε προς επεξεργασία ή ακόμη και για να την παρουσιάσουμε στον χρήστη για να του δώσουμε πληροφορίες. Όπως και σε άλλες γλώσσες προγραμματισμού, έτσι και στην LSL πρέπει να την ορίσουμε πρώτα. Ωστόσο εφόσον καταλαμβάνει μνήμη θέλουμε να φροντίσουμε να έχει περιορισμένη διάρκεια αρκετά συχνά και μόλις ολοκληρώσουμε τα όσα θέλουμε να κάνουμε με αυτήν, να αποδεσμεύσουμε την μνήμη που απαιτείται. Οι μεταβλητές στην LSL έχουν κατασκευαστεί με τέτοιον τρόπο, ώστε να καταστρέφονται μετά το πέρας του τμήματος κώδικα στο οποίο έχουν επιρροή. Μερικές φορές όμως είναι απαραίτητο να διατηρήσουμε την μεταβλητή και την τιμή της καθόλη την διάρκεια του προγράμματος. Σε αυτή την περίπτωση πρέπει να την ορίσουμε ως σφαιρική (global) μεταβλητή στην οποία θα έχουν πρόσβαση όλα τα τμήματα κώδικά μας. Για να ορίσουμε μία μεταβλητή ως σφαιρική αρκεί να την ορίσουμε στην αρχή του προγράμματος μας. Επίσης έχουμε την δυνατότητα να την αρχικοποιήσουμε σε εκάστοτε περίπτωση. Για να ορίσουμε μία μεταβλητή αρκεί να πληκτρολογήσουμε το ακόλουθο:

```
integer myVariable;
```

Κάθε μεταβλητή κατά τον ορισμό της είναι υποχρεωτικό να έχει έναν τύπο, ο οποίος δεσμεύει συγκεκριμένη μνήμη. Στο ανωτέρω παράδειγμα

ο τύπος της μεταβλητής ορίζεται ως integer, δηλαδή ως ακέραιος αριθμός. Η LSL έχει του ακόλουθους τύπους μεταβλητών:

integer myVariable = 4; //Ακέραιος

float myFloat =3.56; // Αριθμός Κινητής Υποδιαστολής

string myString =“This is My Own String”; //Συμβολοσειρά

list words =[“This”,”Is”,”A”,”List”]; // Πίνακας Τιμών

επίσης μια list τυπου μεταβλητή μπορεί να κατέχει ανόμοιου τύπου στοιχεία.

list entries =[“A list may contain many types of values”,2,1.4,<3.0,2.56,0.0>]; // διαφορετικού τύπου μεταβλητές

vector =<1.2,6.1,0.7>; // Διάνυσμα τριών διαστάσεων //ή και τρείς ξεχωριστές μεταβλητές προς ευκολότερη επεξεργασία.

rotation =<1.0,2.0,3.0,4.0>; // Περιστροφή ή τέσσερις //μεταβλητές float τύπου που θέλουμε να έχουμε προσβάσιμες //ευκολότερα.

key =”myString”; // Αλφαριθμητικό ταυτότητα //αντικειμένου ή Avatar

Ιδιαίτερη σημασία πρέπει να δώσουμε στο γεγονός ότι δεν υπάρχουν Boolean τύπου μεταβλητές.Αν θέλουμε να ορίσουμε μεταβλητή ως Boolean πρέπει να την ορίσουμε σαν integer και μετά να της δώσουμε τιμή TRUE ή FALSE όπου μεταγλωτίζεται σωστά από την LSL.

Παραπάνω φαίνεται πως μπορούμε να ορίσουμε μεταβλητή και πώς να την αρχικοποιήσουμε ανάλογα με τον τύπο της. Αν στον κώδικά μας θέλουμε να αλλάξουμε κατά την επεξεργασία τον τύπο των δεδομένων μιας μεταβλητής αρκεί να πληκτρολογήσουμε κάτι παρόμοιο με το ακόλουθο:

```
vartype myTypecast=(vartype)myVariable;
```

όπου vartype είναι ο τύπος δεδομένων στον οποίο θέλουμε να μετατρέψουμε την τιμή της μεταβλητής myVariable.

4.1.5 Βρόγχοι και εντολές συνθηκών διακλάδωσης.

Η LSL μας παρέχει ορισμένες εντολές για έλεγχο ύπαρξης συνθηκών και για εκτέλεση με διακλαδώσεις. Επίσης μας παρέχει εντολές επανάληψης όσο ισχύει κάποια συνθήκη(γνωστές και ως βρόγχοι).

If – else εντολές.

Αν θέλουμε να κάνουμε κάτι να λειτουργήσει μόνο αν κάποια συνθήκη είναι αληθής πρέπει να γράψουμε κάτι παρόμοιο με το ακόλουθο τμήμα κώδικα:

```
if (συνθήκη)
{
```

```
…
}
else if (συνθήκη2)
{
…
}
else
{
…
}
```

Το παραπάνω τμήμα κώδικα εκτελεί τις εντολές μέσα στα άγκιστρα ανάλογα με το ποιά συνθήκη ισχύει και αν δεν ισχύει καμία εκτελείται το τμήμα κώδικα που ακολουθεί το τμήμα else.

Δεν είναι απαραίτητο να υπάρχει τμήμα else ή else if. Παραδείγματος χάρη το τμήμα κώδικα

```
if(συνθήκη)
{
…
}
```

είναι απόλυτα αποδεκτό και μεταφράζεται στο ακόλουθο: Αν ισχύει η συνθήκη τότε εκτέλεσε το τμήμα κώδικα μέσα στα άγκιστρα αλλιώς απλά αγνόησέ το.

For

Ο βρόγχος For χρησιμοποιείται για να εκτελείται ένα τμήμα κώδικα όσο ισχύει μια συνθήκη. Η σύνταξή του είναι η ακόλουθη:

```
for(αρχικοποίηση;συνθήκη;αύξηση)
{
…
}
```

Σε αυτό το τμήμα κώδικα δίνουμε μια τιμή κατά την αρχικοποίηση σε μία μεταβλητή ελέγχου , μία συνθήκη που πρέπει να ελέγχεται κατά την εκτέλεση του κώδικα κάθε φορά και μία τιμή κατά την οποία θα αυξάνει η θα μειώνεται η μεταβλητή την οποία θα ελέγχουμε κατά την συνθήκη.

While

Το ίδιο αποτέλεσμα θα έχει και ο βρόγχος while αλλά έχει διαφορετική σύνταξη.

```
while(συνθήκη)
{
Εντολές προς εκτέλεση;
Προσαύξηση;
}
```

Do – While

Τέλος αν θέλουμε κάτι να εκτελεστεί τουλάχιστον μία φορά αρκεί να χρησιμοποιήσουμε τον βρόγχο do-while που έχει την ακόλουθη σύνταξη:

do

{

Εντολές προς εκτέλεση;

}

while (συνθήκη)

4.1.6 Μέθοδοι

Οι μέθοδοι είναι τμήματα κώδικα που είτε είναι προκατασκευασμένα ή τα κατασκευάζουμε εμείς για να εξυπηρετήσουμε τις ανάγκες μας. Μια μέθοδος μπορεί είτε να ανήκει σε κάποια κατάσταση στην οποία βρίσκεται το αντικείμενο για το οποίο γράφουμε τον κώδικα ή μπορεί να είναι σφαιρική όπως ακριβώς γίνεται και με τις μεταβλητές. Έτσι αν ορίσουμε μια μέθοδο στην αρχή του προγράμματος , τότε θα είναι προσβάσιμη από οποιαδήποτε κατάσταση ενώ αν την ορίσουμε μέσα στην εμβέλεια κάποιας κατάστασης τότε θα είναι τοπική και θα λειτουργεί μόνο εντός της εμβέλειάς της. Όσες μέθοδοι ξεκινούν με το ll μπροστά ανήκουν στις προκατασκευασμένες μεθόδους του LSL compiler και είναι σφαιρικές.

Μία μέθοδος είναι δυνατόν να απαιτεί παραμέτρους κατά την δήλωσή της, οι οποίες θα είναι απαραίτητες για την λειτουργία της. Ο ορισμός μιας μεθόδου γίνεται με τον ακόλουθο τρόπο:

```
MyMethod(τύπος_παραμετρου1 παράμετρος1, …τύπος_παραμέτρουN παράμετροςN)
{
…
}
```

Τα ονόματα των παραμέτρων κατά την δήλωση μιας μεθόδου δεν είναι απαραίτητα ίδια με τα ονόματα των μεταβλητών που θα χρησιμοποιήσουμε κατά την κλήση της μεθόδου. Επίσης αν θέλουμε η μέθοδός μας να επιστρέφει κάποια τιμή τότε πρέπει να προσθέσουμε στο τέλος του κώδικά μας την εντολή return ακολουθούμενη από την τιμή που θέλουμε να επιστρέφεται από την μέθοδό μας και το σύμβολο ;

Τέλος αν θέλουμε να καλέσουμε μια συνάρτηση που έχουμε ορίσει πληκτρολογούμε το όνομά της και τις παραμέτρους που απαιτεί και βάζουμε το σύμβολο ; στο τέλος.

```
MyMethod(παράμετρος1, … παράμετροςN);
```

4.2 Δημιουργία ενός προγράμματος (script)

Στα πλαίσια τις ολοκληρωμένης κάλυψης της γλώσσας LSL και OSSL θα κατασκευάσουμε ορισμένα προγράμματα προς επίδειξη των δυνατοτήτων τους. Έτσι θα πρέπει να εξηγήσουμε αρχικά πως προσθέτουμε ένα script σε κάποιο αντικείμενο. Ας ληφθεί υπόψη ότι αν ένα αντικείμενο στο χώρο είναι σύνθετο το script που θα προστεθεί, προστίθεται πάντα στο αντικείμενο ρίζα (root).

Για να προσθέσουμε ένα script σε κάποιο αντικείμενο πατάμε το δεξί πλήκτρο του ποντικιού πάνω στο αντικείμενο και επιλέγουμε Edit. Μετά επιλέγουμε την καρτέλα content και πατάμε στην επιλογή New Script. Τέλος πατάμε δύο φορές πάνω στο New Script που προστέθηκε για να το

ανοίξουμε και να κάνουμε όποια αλλαγή θέλουμε. Αρχικά το script που προστίθεται είναι πάντα το ίδιο και είναι το ακόλουθο:

```
default
{
   state_entry()
   {
      llSay(0, "Script running");
   }
}
```

Το ανωτέρω πρόγραμμα κατά την έναρξή του , κάνει το αντικείμενο στο οποίο είναι προσκολλημένο να χρησιμοποιήσει το Local Chat (τοπικό chat room) και να πει (say) την φράση Script running.

Αρχικά έχουμε την προκαθορισμένη αρχική κατάσταση που κάθε πρόγραμμα υποχρεωτικά έχει, την default. Αμέσως μετά ορίζουμε ένα συμβάν (event) , το state_entry το οποίο περιέχει ένα τμήμα κώδικα προς εκτέλεση. Τα συμβάντα είναι πιθανόν να παίρνουν παραμέτρους όπως ακριβώς και οι μέθοδοι. Έτσι το συμβάν state_entry βλέπουμε ότι κατά την σύνταξή του ακολουθείται από (). Το state_entry ειδικά δεν παίρνει καμία παράμετρο. Αυτό που ορίζει το συγκεκριμένο συμβάν είναι ότι κάθε φορά που μπαίνουμε σε αυτήν την κατάσταση το τμήμα κώδικα που περικλύει θα εκτελείται.

Τέλος η εντολή llSay(integer channel, string message); όπως είναι συντεταγμένη , κάνει το prim να χρησιμοποιήσει το τοπικό chat και να πει Script running.

4.3 Βασικά συμβάντα

Τα συμβάντα, όπως προαναφέρθηκε, αποτελούν τα δεδομένα εισόδου που η μηχανή καταστάσεων που φτιάχνουμε χρησιμοποιεί. Στόχος είναι να καλυφθούν τα πλέον βασικά ώστε να μπορούμε να αναπτύξουμε εφαρμογές.

state_entry() {

…}

Όπως αναφέρθηκε νωρίτερα το συμβάν state_entry εκτελείται κάθε φορά που μεταφερόμαστε στην κατάσταση που το εμπεριέχει. Σκοπός ύπαρξής του είναι ο προσδιορισμός των συνθηκών που επικρατούν στην κατάσταση στην οποία ανήκει.

state_exit() {

…}

Παρόμοια με την state_entry, η state_exit ορίζει τι θα συμβεί όταν το σύστημα μεταβαίνει σε κάποια άλλη κατάσταση. Όπως βλέπουμε ούτε η state_exit δέχεται παραμέτρους.

touch_start (integer num_detected) {

…}

Το συμβάν touch_start ορίζει τι συμβαίνει την στιγμή που ‘’ακουμπάμε’’ το αντικείμενο για το οποίο γράφεται το script ενώ βρίσκεται στην κατάσταση που περιέχει το touch_start συμβάν. Για να ακουμπήσουμε το αντικείμενο πρέπει απλά να πατήσουμε το αριστερό πλήκτρο του ποντικιού ενώ ο δείκτης του βρίσκεται πάνω στο αντικείμενο. Η παράμετρος num_detected ορίζεται ως ακέραιος και αποθηκεύει τον αριθμό των χρηστών που άγγιξαν το αντικείμενο στον τελευταίο κύκλο ρολογιού του υπολογιστή.

Αν θέλουμε να εκτελείται το τμήμα κώδικα όσο εμείς συνεχίζουμε να αγγίζουμε το αντικείμενο αντί για το συμβάν touch_start, χρησιμοποιούμε το touch(integer num_detected) ενώ αν θέλουμε κάτι να συμβεί μόλις σταματήσουμε να αγγίζουμε το αντικείμενο , χρησιμοποιούμε το touch_end(integer num_detected)

timer() {

…}

Με το συμβάν timer μπορούμε να ορίσουμε κάποιο χρονικό συμβάν. Παραδείγματος χάρη αν θέλουμε ένα αντικείμενο να αλλάζει χρώμα κάθε μερικά δευτερόλεπτα πρέπει να χρησιμοποιήσουμε ένα timer συμβάν. Το timer συμβάν εκτελείται κάθε κάποιον αριθμό δευτερολέπτων που ορίζουμε εμείς με την μέθοδο llSetTimerEvent(float timeinseconds). Ακολουθεί ένα απλό τμήμα κώδικα για να μας βοηθήσει να κατανοήσουμε ένα timer event.

```
default
{
   touch_start(integer num)
   {
      llSetTimerEvent(1.0);
   }
   touch_end(integer num)
   {
      llSetColor(<1.0,0.0,0.0>,ALL_SIDES);
      llSetTimerEvent(0);
   }
   timer()
   {
      llSetColor(<0.0,1.0,0.0>,ALL_SIDES);
   }
}
```

Για την κατανόηση του script αναφέρεται ότι η εντολή llSetColor αλλάζει το χρώμα του αντικειμένου μας. Αυτή η εντολή θα επεξηγηθεί αργότερα πιο αναλυτικά.

Αρχικά βλέπουμε τον ορισμό της default καταστάσεως. Αμέσως μετά ορίζεται ένα συμβάν, το touch_start και μέσα σε αυτό με την εντολή llSetTimerEvent(1.0) εκκινεί ένα χρονόμετρο το οποίο κάθε ένα δευτερόλεπτο θα πραγματοποιεί ένα συμβάν timer. Μετά από αυτό ορίζεται το συμβάν touch_end το οποίο ορίζει ότι την στιγμή που θα σταματήσουμε να αγγίζουμε το αντικείμενο , το χρώμα του αντικειμένου πρέπει να γίνει κόκκινο. Επίσης με το llSetTimerEvent(0) προκαλούμε την παύση και τον μηδενισμό οποιουδήποτε χρονομέτρου λειτουργεί εκείνη την στιγμή. Τέλος ορίζεται το timer συμβάν το οποίο προκαλεί την αλλαγή του χρώματος του αντικειμένου σε πράσινο.

Η συμπεριφορά του ανωτέρου προγράμματος είναι ότι αν αγγίζουμε το αντικείμενο για παραπάνω από ένα δευτερόλεπτο το αντικείμενο θα γίνει πράσινο για όση ώρα συνεχίσουμε να το αγγίζουμε και θα γίνει κόκκινο όταν σταματήσουμε.

sensor(integer num) και no_sensor()

Το συμβάν sensor εκτελείται όταν σε έλεγχο που γίνεται γύρω από το αντικείμενο βρεθεί είτε κάποιο άλλο αντικείμενο ή κάποια ενσάρκωση. Ο έλεγχος αυτός μπορεί να γίνει μία φορά ή να γίνεται κάθε κάποια τακτά χρονικά διαστήματα που εμείς ορίζουμε. Για να προκαλέσουμε έναν τέτοιο έλεγχο , χρησιμοποιούμε τις μεθόδους llSensor(string name,key id,integer type,float range, float arc) ή llSensorRepeat(string name,key id,integer type,float range, float arc, float time.Το string name είναι το όνομα κάποιας ενσάρκωσης ή αντικειμένου για την οποία θέλουμε να κάνουμε έλεγχο κατά πόσο είναι κοντά. Με key id ορίζουμε αν ψάχνουμε για κάποιο αντικείμενο με συγκεκριμένο UUID στο χώρο. Το integer type ουσιαστικά καθορίζει αν κάνουμε έλεγχο για ενσάρκωση ή αντικείμενο καθώς επίσης και το αν περιέχει κάποιο script ή όχι. Αποδεκτές τιμές για το type είναι οι:

AGENT: Ελέγχει για χρήστες

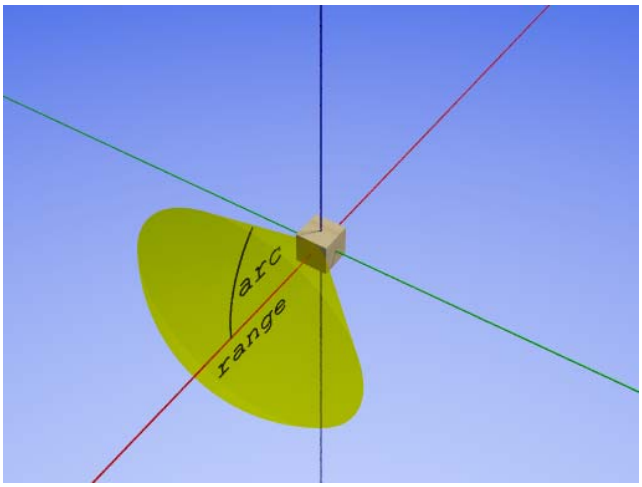
ACTIVE: Ελέγχει για κινούμενα αντικείμενα, αντικείμενα τύπου physical ή ενσαρκώσεις καθώς επίσης και αντικείμενο πού έχουν κάποιο script ενεργό

PASSIVE: Ελέγχει για ακίνητα αντικείμενα, non physical αντικείμενα ή για αντικείμενα με script το οποίο δεν είναι ενεργό εκείνη της χρονική στιγμή .Το float range αντιπροσωπεύει την εμβέλεια του ελέγχου. Εν συνεχεία το float arc εκφράζει την γωνία στην οποία γίνεται έλεγχος σε ακτίνια(radians) γύρω από τον άξονα x'x. Χρήσιμες τιμές που αναγνωρίζονται από τον LSL compiler είναι PI , PI_BY_TWO καθώς επίσης και οι υποδιαιρέσεις τους.

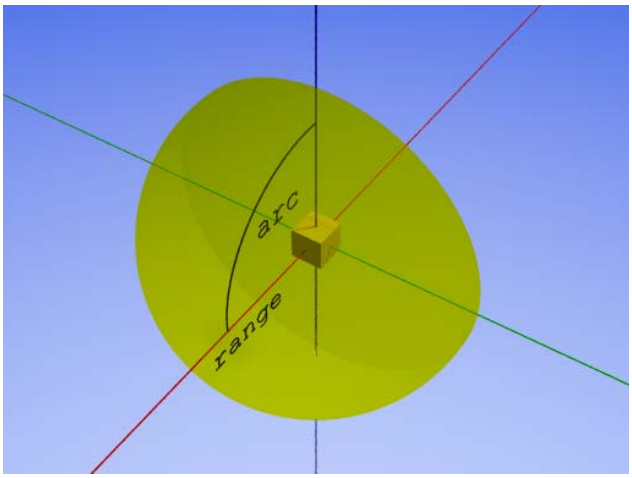
SCRIPTED: Ελέγχει για αντικείμενα με ενεργό script την χρονική στιγμή του ελέγχου.

Επίσης μπορούμε να τα συνδυάσουμε με Bitwise – OR που συμβολίζεται με | στην LSL.

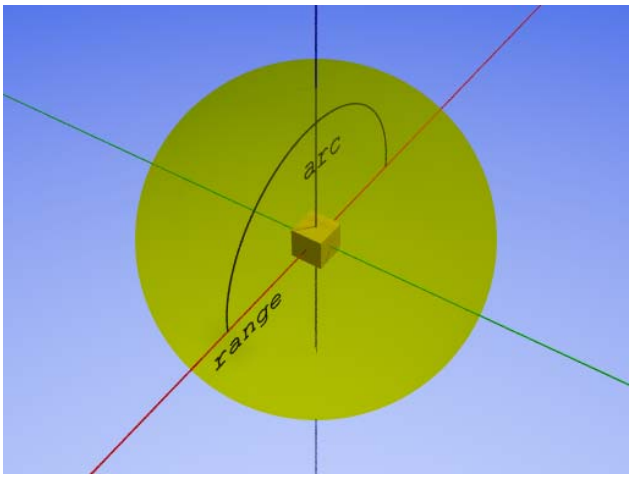
PI/4



PI_BY_TWO



PI



Τέλος με float time παράμετρο στην llSensorRepeat ορίζουμε πόσο συχνά θέλουμε να γίνεται ένας έλεγχος. Αν θέλουμε να σταματήσουμε τους διαρκείς ελέγχους τότε απλά πληκτρολογούμε στο σημείο του προγράμματος που θέλουμε να σταματάει ο έλεγχος, llSensorRemove() με την οποία επίσης καταστρέφουμε και όλα τα δεδομένα τον προηγούμενων ελέγχων αν υπάρχουν.

Για να ορίσουμε ένα sensor συμβάν πρέπει να πληκτρολογήσουμε κάτι παρόμοιο με

sensor(integer num_detected) {

…}

Το integer num_detected αντιπροσωπεύει των αριθμό των αντικειμένων, ενσαρκώσεων που βρέθηκαν. Ομοίως για να ορίσουμε κάποιο συμβάν αν ο έλεγχός μας δεν φέρει αποτέλεσμα εύρεσης πρέπει να χρησιμοποιηθεί η ακόλουθη μέθοδος ‘’no_sensor()’’

Για την κατανόηση παρατίθεται κώδικας και γίνεται επεξήγησή του παρακάτω.

```
default
{
  state_entry()
  {
    llSetColor(<1.0,0.0,0.0>,ALL_SIDES);
    llSensorRepeat( "",NULL_KEY,AGENT, 10.0, PI ,0.5);
  }
    sensor(integer num)
  {
   state green;
  }
}

state green
{
  state_entry()
  {
    llSetColor(<0.0,1.0,0.0>,ALL_SIDES);
    llSensorRepeat( "",NULL_KEY,AGENT, 10.0, PI ,0.5);
  }
  no_sensor()
  {
    state default;
  }
}
```

Αρχικά ορίζουμε την default κατάσταση στην οποία έχουμε δύο τύπων συμβάντα. Έχουμε το state_entry στο οποίο ορίζουμε ότι το χρώμα του αντικειμένου μας θα είναι κόκκινο μέσω της llSetColor. Επίσης ορίζουμε στο αντικείμενό μας να κάνει έλεγχο για avatars σε λιγότερο από 15 μέτρα από αυτό με την εντολή llSensorRepeat. Αμέσως μετά ορίζουμε ότι το αντικείμενο πρέπει να αλλάζει κατάσταση όταν ο έλεγχος μας αποφέρει μη μηδενικό αποτέλεσμα και να μεταβαίνει στην κατάσταση green. Μετά ορίζουμε την κατάσταση green στην οποία έχουμε πάλι δύο μόνο συμβάντα. Έχουμε state_entry στο οποίο ορίζουμε ότι το χρώμα του αντικειμένου μας πρέπει να είναι πράσινο και ότι πρέπει να γίνεται έλεγχος για avatars κάθε 0.5 δευτερόλεπτα. Εδώ ορίζουμε ένα νέο συμβάν το no_sensor το οποίο ,αν δεν βρεθεί κανένας agent κατά τον έλεγχο τότε το αντικείμενο επιστρέφει στην κατάσταση default.

listen(integer channel, string objectname, key id, string message)

Τα συμβάντα listen είναι ίσως τα πιο χρήσιμα για να προκαλέσουμε κάποια μεταβολή σε κάποιο αντικείμενο καθώς μας επιτρέπουν να προκαλέσουμε την αλλαγή απλά πληκτρολογώντας κάτι σε συγκεκριμένο κανάλι. Τα συμβάντα listen επίσης είναι δυνατόν να προκαλούνται μόνο από κάποιον συγκεκριμένο χρήστη ή αντικείμενο. Έτσι παραδείγματος χάρη, όπως θα δούμε σύντομα , έχουμε την δυνατότητα ακουμπώντας ένα αντικείμενο να προκαλέσουμε μεταβολή σε παραπάνω από ένα.

Ένα συμβάν listen προκαλείται όταν το σωστό μήνυμα δοθεί από συγκεκριμένο αντικείμενο ή ενσάρκωση στο σωστό κανάλι. Ο ορισμός ενός συμβάντος listen γίνεται με τον ακόλουθο τρόπο.

```
listen(integer channel, string objectname, key id, string message)
{
…
}
```

Οι παράμετροι του συμβάντος είναι οι ακόλουθες:

Integer channel: Αντιπροσωπεύει το κανάλι στο οποίο το αντικείμενο αναμένει κάποιο μήνυμα. Το κανάλι 0 είναι το κανάλι που χρησιμοποιούν όλοι οι χρήστες για να συνομιλήσουν με άλλους χρήστες.

String objectname: Αντιπροσωπεύει το όνομα του αντικειμένου από το οποίο αναμένει το μήνυμα. Αν αντί για μήνυμα χρησιμοποιήσουμε “” τότε οποιοδήποτε αντικείμενο που θα χρησιμοποιήσει το κανάλι αυτό για

να στείλει το συγκεκριμένο μήνυμα είναι δυνατόν να προκαλέσει το συμβάν.

Key id: Αντιπροσωπεύει το UIID (μοναδικό κλειδί) κάποιου συγκεκριμένου αντικειμένου ή ενσάρκωσης από το οποίο αναμένεται το μήνυμα. Αν αντί για μήνυμα βάλουμε “” τότε οποιοδήποτε αντικείμενο ή ενσάρκωση που θα χρησιμοποιήσει το κανάλι αυτό για να στείλει το συγκεκριμένο μήνυμα είναι δυνατόν να προκαλέσει το συμβάν.

String message: Μέσα σε εισαγωγικά εδώ πρέπει να βρίσκεται το συγκεκριμένο μήνυμα το οποίο αναμένεται για να προκληθεί το listen συμβάν. Αν αντί για μήνυμα βάλουμε “” τότε οποιοδήποτε μήνυμα θα προκαλεί ένα listen συμβάν.

Η μέθοδος που ορίζει ότι ένα αντικείμενο αρχίζει να αναμένει κάποιο μήνυμα είναι η llListen. Το παρακάτω παράδειγμα μας εξηγεί πώς λειτουργεί ένα listen συμβάν. Για λόγους απλότητας θα χρησιμοποιηθούν κυρίως εντολές που έχουμε δει ως τώρα…

```
default
{
  state_entry()
  {
    llSetColor(<1.0,0.0,0.0>,ALL_SIDES);                    //κάνει το αντικείμενο
                                                            //κόκκινο
    llListen(44,"","","green" );                            //αναμένει το μήνυμα
                                                            //green στο κανάλι 44

  }
    listen(integer chan,string name,key id,string msg)
  {
                state green;                                //μόλις λάβει το σωστό
                                                            //μήνυμα μεταβαίνει στην
                                                            //κατάσταση green.
  }
}

state green
{
  state_entry()
  {
```

```
    llSetColor(<0.0,1.0,0.0>,ALL_SIDES);                    //κάνει το αντικείμενο
                                                            //πράσινο.

    llListen(44,"","","red" );                              //αναμένει το μήνυμα
                                                            //red στο κανάλι 44
  }
    listen(integer chan,string name,key id,string msg)
  {
   state default;                                           //μόλις λάβει το σωστό
                                                            //μήνυμα μεταβαίνει στην
                                                             //προεπιλεγμένη default
                                                            //κατάσταση.

  }
}
```

Το παραπάνω τμήμα κώδικα ως σκοπό έχει να κάνει το αντικείμενο να αναμένει είτε το μήνυμα green είτε το red στο κανάλι 44 από οποιονδήποτε και να ορίζει το χρώμα του αντικειμένου στο οποίο το έχουμε προσθέσει σε πράσινο ή κόκκινο αντίστοιχα.

Αρχικά ορίζουμε την κατάσταση default όπου κατά την είσοδό μας σε αυτήν το αντικείμενο παίρνει το κόκκινο χρώμα. Εκεί ορίζουμε ότι το αντικείμενο πρέπει να αναμένει στο κανάλι 44 από οποιονδήποτε ανεξαρτήτως ονόματος και key id το μήνυμα green. Μετά ορίζουμε ένα συμβάν listen στο οποίο ορίζουμε τι θα γίνει αν το αντικείμενο δεχτεί το μήνυμα που αναμένει , όπου στην περίπτωσή μας ορίζουμε να μεταβεί στην κατάσταση green.

Αμέσως μετά ορίζουμε την κατάσταση green, όπου κατά την είσοδό μας σε αυτήν , το αντικείμενο γίνεται πράσινο. Επίσης το αντικείμενο τίθεται σε αναμονή για το μήνυμα red στο κανάλι 44 πάλι αποδεκτό από οποιονδήποτε.Τέλος ορίζουμε ότι αν το αντικείμενο ενώ βρίσκεται στην κατάσταση green δεχτεί το μήνυμα red τότε επιστρέφει στην προκαθορισμένη default κατάσταση.

Μια καλή πρακτική αν θέλουμε να ορίσουμε παραπάνω από ένα πιθανά μηνύματα τα οποία το αντικείμενό μας θέλουμε να συλλαμβάνει όσο βρίσκεται σε μία κατάσταση είναι να χρησιμοποιούμε κατά την κλήση της llListen “” στο τμήμα που αντιστοιχεί στο μήνυμα το οποίο αναμένουμε και στη συνέχεια να ελέγχουμε με εμφωλευμένα if το μήνυμα το οποίο έχουμε λάβει και να κάνουμε διαφορετικά πράγματα ανάλογα το μήνυμα. Ένα μικρό παράδειγμα χωρίς επεξήγηση ακολουθεί.

```
default
{
   state_entry()
   {
      llSetColor(<1.0,0.0,0.0>,ALL_SIDES);
      llListen(45,"","","" );
   }
   // Όταν αγγίζουμε το αντικέιμενο μας παρέχονται πληροφορίες για
   //πιθανά commands και σε πιο κανάλι επικοινωνίας να τα δώσουμε
   touch_start(integer num)
   {
      llSay(0,"Options:\ngreen\nyellow\nblue\ncyan\nred\nChannel number :45");
   }
   // Ελέγχει τι είπε ο χρήστης και δρα ανάλογα.
   listen(integer chan,string name,key id,string msg)
   {
      if (msg=="green")
      {
         llSetColor(<0.0,1.0,0.0>,ALL_SIDES);
      }
      else if (msg=="yellow")
      {
         llSetColor(<1.0,1.0,0.0>,ALL_SIDES);
      }
      else if(msg=="blue")
      {
         llSetColor(<0.0,0.0,1.0>,ALL_SIDES);
      }
      else if(msg=="cyan")
      {
         llSetColor(<0.0,1.0,1.0>,ALL_SIDES);
      }
      else if(msg=="red")
      {
         llSetColor(<1.0,0.0,0.0>,ALL_SIDES);
      }
      else
      {
         llSay(0,"Not a valid command");
      }
   }
}
```

5.Προγραμματίζοντας σε LSL.

5.1 Εντολές προς κανάλια συνομιλίας και εντολές εμφάνισης κειμένου .

Προαναφέρθηκε νωρίτερα ότι υπάρχουν πολλά κανάλια συνομιλίας στο Second Life. Το κανάλι 0 είναι το κανάλι που ακούει η ενσάρκωσή μας. Η χρησιμότητα των υπολοίπων καναλιών βρίσκεται στο γεγονός ότι μπορούμε να ορίσουμε ένα listen συμβάν χρησιμοποιώντας κάποιο άλλο κανάλι. Ωστόσο υπάρχουν φορές που θέλουμε κάποιο αντικείμενο να λέει κάτι το οποίο θα χρησιμοποιήσουμε για να εκκινήσουμε συμβάντα σε άλλα αντικείμενα.

Παράδειγμα αποτελεί μια διπλή πόρτα που εμείς θέλουμε να ανοίγει και τα δυο φύλα αυτόματα όταν χρησιμοποιούμε οποιοδήποτε από τα δύο. Για να γίνει το παραπάνω χωρίς να υπάρχουν καθυστερήσεις πρέπει να φτιάξουμε 3 αντικείμενα. Τα δυο φύλα της πόρτας τα οποία θα αναμένουν κάποιο συγκεκριμένο μήνυμα σε κάποιο συγκεκριμένο κανάλι και ένα πλήρως διάφανο και phantom αντικείμενο που θα έχει τις διαστάσεις της πόρτας και που όταν το ακουμπάμε στέλνει το μήνυμα που θέλουμε στις πόρτες ώστε να ανοίξουν. Για να γίνει το τελευταίο πρέπει στο touch_start συμβάν που θα χρησιμοποιήσουμε να προσθέσουμε μία εντολή llSay ή llWhisper ή llShout.

Η σύνταξη και οι παράμετροι που παίρνουν οι ανωτέρω εντολές είναι ίδιες και για τις τρεις και είναι οι ακόλουθες.

llSay(integer channel, string message);

llWhisper(integer channel, string message);

llShout(integer channel, string message);

Για να κατανοήσουμε τις εντολές ας δούμε το παρακάτω παράδειγμα. Όταν πρωτοκατασκευάζουμε ένα script το αντικείμενο αποκτά ένα default script. Το τμήμα κώδικα είναι το ακόλουθο:

```
default
{
  state_entry()
  {
```

```
    llSay(0, "Script running");
  }
}
```

Η επεξήγηση του ανωτέρω προγράμματος έχει γίνει και νωρίτερα. Η llSay όπως είναι συντεταγμένη ορίζει ότι το αντικείμενο θα χρησιμοποιήσει το κανάλι 0 και θα εμφανίσει εκεί το μήνυμα Script running. Αλλάζοντας σε llShout και llWhisper επιρρεάζουμε την εμβέλεια του μηνύματος. Για παράδειγμα αν αλλάζαμε το llSay σε llWhisper και το αντικείμενο βρισκόταν παραπάνω από 1 m μακριά από εμάς τότε εμείς δεν θα βλέπαμε το μήνυμα Script running.

Πέρα από αυτό , έχουμε την δυνατότητα να προσθέσουμε κείμενο σε κάποιο αντικείμενο για να το κάνουμε ορατό ως αντικείμενο άξιο προσοχής. Όταν η ενσάρκωσή μας θα βρεθεί εντός κάποιας ακτίνας από το αντικείμενο το κείμενο θα εμφανιστεί από πάνω του. Η μέθοδος που παρέχει αυτή την δυνατότητα είναι η llSetText(string msg,vector color,float alpha). Το ακόλουθο script χρησιμοποιεί την llSetText για να παρουσιάσει το μήνυμα click me πάνω από το αντικείμενο.

```
default
{
  state_entry()
  {
    llSetText("click me",<0.,0.,0.>,.5);          // προσθήκη του κειμένου
                                                  //click me πάνω  από
                                                  // το αντικείμενο
  }
}
```

Για λόγους ευκολίας κατανόησης απλά θα αναφερθεί ότι το διάνυσμα <0.,0.,0.> αντιπροσωπεύει το μαύρο χρώμα και ότι η τιμή του alpha αντιπροσωπεύει πόσο διαφανές θα είναι το κείμενο. Περισσότερα για το alpha και τα διανύσματα χρωμάτων θα αναφερθούν παρακάτω.

5.2 Μέθοδοι αλλαγής παραμέτρων αντικειμένων

Το χρώμα , η φωτεινότητα, η διαφάνεια, οι διαστάσεις, η κλίση και άλλα στοιχεία ορίζουν ένα αντικείμενο στο χώρο. Μεγέθη σαν αυτά αποτελούν τις παραμέτρους ενός αντικειμένου. Κατά συνέπεια είναι απολύτως λογικό να επιθυμεί ένας χρήστης να μπορεί να τα επηρεάσει ανάλογα με τις ανάγκες του. Η LSL μας παρέχει μεθόδους για αυτές τις περιπτώσεις οι οποίες μας προσφέρουν απόλυτο έλεγχο.

5.2.1 Αλλαγή διαστάσεων αντικειμένου

Οι διαστάσεις ενός αντικειμένου κατά την δημιουργία του ορίζονται μέσω τριών αριθμών. Έναν για την διάσταση x(μήκος), έναν για την διάσταση y(πλάτος) και έναν για την διάσταση z (ύψος - βάθος). Μία μέθοδος που μπορούμε να χρησιμοποιήσουμε για να επαναπροσδιορίσουμε τις διαστάσεις ενός αντικειμένου μέσω script είναι η llSetScale.

llSetScale(vector dimensions);

Η παραπάνω μέθοδος παίρνει σαν παράμετρο ένα διάνυσμα το οποίο περιέχει σε κάθε ένα από τα τρία χαρακτηριστικά του, την τιμή μιας εκ των επιθυμητών διαστάσεων.

dimensions.x = διάσταση x
dimensions.y = διάσταση y
dimensions.z = διάσταση z

Οι τιμές που μπορεί να πάρει το x , y , z είναι από 0.001 έως 256 ή 64 αν επιτρέπονται ή όχι τα megaprims στον Server που χρησιμοποιούμε αντίστοιχα. Στον OpenSimulator τα megaprims επιτρέπονται. Το ακόλουθο script μετατρέπει ένα κυβικό σχήμα οποιονδήποτε διαστάσεων σε ένα ορθογώνιο παραλληλεπίπεδο με διαστάσεις <5.0,2.0,2.0> και μετά από 5 δευτερόλεπτα το μετατρέπει σε έναν κύβο με διαστάσεις <0.5,0.5,0,5>.

```
default
{
   touch_start(integer num)              //αλλαγή διαστάσεων και έναρξη timer
   {
      llSetScale(<5.,2.,2.>);
      llSetTimerEvent(5.0);
   }
   timer()                               //επιστροφή στον default κύβο και
                                         //ακύρωση του timer
   {
      llSetScale(<0.5,0.5,0.5>);
      llSetTimerEvent(0.0);
   }
}
```

5.2.2 Θέση και μετατόπιση

Κάθε αντικείμενο στο χώρο καταλαμβάνει κάποια θέση. Έχει κάποιες συντεταγμένες και με βάση αυτές μπορούμε να επηρεάσουμε τόσο τον χώρο όσο και το ίδιο το αντικείμενο. Αν θέλουμε να βρούμε την τρέχουσα θέση του αντικειμένου ή σύνθετου αντικειμένου στο οποίο προσκολλούμε ένα script αρκεί να χρησιμοποιήσουμε την μέθοδο

llGetPos( ) ;

Η llGetPos( ) επιστρέφει σε μορφή διανύσματος τις συντεταγμένες του αντικειμένου που περιέχει τον κώδικα. Επιπλέον με χρήση αυτών των συντεταγμένων και την εντολή llSetPos(vector pos) μπορούμε να μεταβάλουμε την θέση ενός αντικειμένου που δεν είναι ορισμένο ως physical δηλαδή χωρίς την χρήση δυνάμεων φυσικής. Με κάθε κλήση της μεθόδου το αντικείμενο μπορεί να μετατοπιστεί μέχρι 10m. Επίσης άθε φορά που καλείται η llSetPos(vector pos) κάνει το script ανενεργό για 0.2 δευτερόλεπτα οπότε αν θέλουμε να κάνουμε πολλαπλές μετακινήσεις είναι καλύτερα να χρησιμοποιήσουμε την llSetPrimitiveParams μέθοδο την οποία θα δούμε αργότερα.

Στο πρόγραμμα που ακολουθεί γίνεται μια μικρή επίδειξη ορισμένων χαρακτηριστικών της LSL με χρήση της llGetPos() και της llSetPos(vector pos):

```
vector LIFT_TOP    = <226.,182.,36.157>;
```

```
vector LIFT_BOTTOM = <226.,182.,26.157>;

Travelator(vector start, vector finish)              //Ορίζουμε μια συνάρτηση μετατόπισης
    {
      float  distance = 0;                           //Ορίζουμε την μεταβλητή distance
      float  progress = 0;                           //Ως progress ορίζουμε την πρόοδο
      float increment = 1;                           //Ορίζουμε την προσαύξηση

      vector direction = finish - start;             //Βρίσκουμε την κατεύθυνση κίνησης
      vector dir_normal = llVecNorm(direction);          //και την κανονικοποιούμε
      distance = llVecDist(finish, start);           //Βρίσκουμε την διαφορά των
                                                     //δύο διανυσμάτων

    vector next_pos = start;                         //Αρχικοποιούμε και ορίζουμε την next_pos
                                                     //να ισούται αρχικά με την αρχική μας θέση

    do                                               //Βρόγχος ομαλής μετακίνησης.
      {
        progress += increment;
        if (progress >= distance)
        {
          llSetPos(finish);
        }
        next_pos += increment * dir_normal;
        llSetPos(next_pos);
      } while (progress < distance);
    }

    default                                          //ορισμός στην default κατάσταση
    {                                                //με άγγιγμα μετατόπιση στην LIFT_TOP
      touch_start(integer num_detected)
      {
        Travelator(llGetPos(), LIFT_TOP);
        state top;
      }
    }

    state top                                        //ορισμός στην top κατάσταση
    {                                       //με άγγιγμα μετατόπιση στην LIFT_BOTTOM
      touch_start(integer num_detected)
      {
        Travelator(llGetPos(), LIFT_BOTTOM);
        state default;
      }
    }
```

Στο παραπάνω script κατασκευάζουμε μια μέθοδο την Travelator η οποία παίρνει σαν όρισμα την αρχική θέση και την θέση στην οποία θέλουμε το αντικείμενό μας να καταλήξει. Μέσα στο σώμα της μεθόδου ορίζουμε τρεις μεταβλητές. Αυτές είναι οι…

distance – πρόκειται για την απόσταση μεταξύ της τρέχουσας κατάστασης και του προορισμού.

progress – πρόκειται για την απόσταση που εχει διανυθεί κατά την διάρκεια της εκτέλεσης της μεθόδου.

increment – πρόκειται για την μεταβολή στην θέση του αντικειμένου ανά βήμα.

Αμέσως μετά βρίσκουμε την διεύθυνση της κίνησης αφαιρώντας από το διάνυσμα τελικής θέσης την αρχική και μετά κανονικοποιώντας το διάνυσμα που προκύπτει. Για την κανονικοποίηση χρησιμοποιείται η μέθοδος llVecNorm(vector v1). Τέλος υπολογίζουμε την απόσταση μεταξύ του διανύσματος επιθυμητής θέσης και του διανύσματος αρχικής θέσης ως το μέτρο του διανύσματος που προκύπτει από την αφαίρεση τους μέσω της μεθόδου llVecDist(vector v2,vector v3) και ορίζουμε ότι η μεταβλητή next_pos θα πρέπει να πάρει την τιμή της αρχικής θέσης (αρχικοποιούμε την μεταβλητή).

Ακολουθεί ένας βρόγχος στον οποίο όσο η απόσταση που το αντικείμενο έχει διανύσει είναι μικρότερη από την απόσταση της αρχικής θέσης από την τελική , τότε το αντικείμενο μετατοπίζεται κατά increment προς την κατεύθυνση που οδηγεί άμεσα στην τελική επιθυμητή θέση. Στην αρχή του μπλοκ κώδικα του βρόγχου η μεταβλητή progress προσαυξάνεται κατά increment και αμέσως μετά γίνεται έλεγχος αν η progress είναι μεγαλύτερη η ίση από την distance και αν είναι το αντικείμενο απλά μετατοπίζεται στην τελική θέση.

Τέλος ορίζονται δύο καταστάσεις όπου στην default με touch εκτελείται η Travelator με ορίσματα την τρέχουσα θέση του αντικειμένου και την τελική θέση και μετά η κατάσταση αλλάζει σε top και στην top όπου πάλι καλείται η Travelator με ορίσματα την τρέχουσα θέση του αντικειμένου και την αρχική του θέση και επιστροφή στην default κατάσταση. Οι μεταβλητές LIFT_TOP,LIFT_BOTTOM που ορίζονται αρχικά στο πρόγραμμα υπάρχουν κυρίως για κατανόηση του τι αντιπροσωπεύουν οι τιμές που παίρνει η Travelator σε κάθε κλήση της ενώ για βελτιστοποίηση της απόδοσης από πλευράς μνήμης θα μπορούσαν να παραληφθούν και να χρησιμοποιηθούν οι πραγμάτικες τιμές κατά την κλήση της μεθόδου κάθε φορά.

5.2.3 Περιστροφή αντικειμένου γύρω από κάποιον άξονα και χρήσιμες συναρτήσεις.

Η LSL όπως είδαμε παρέχει ξεχωριστό τύπο για μεταβλητές τύπου rotation(περιστροφής) ο οποίος αντιπροσωπεύει ένα quaternion (τετραδόνιο) καθώς είναι ο πιο αποτελεσματικός τρόπος αναπαράστασης περιστροφών επιπέδου γύρω από κάποιον ορισμένο άξονα και διότι αποτρέπει την εμφάνιση του φαινομένου gimbal lock(http://www.youtube.com/watch?v=rrUCBOlJdt4). Ένα LSL quaternion λοιπόν είναι ας φανταστούμε για απλοποίηση ένα σύνολο από ένα τρισδιάστατο διάνυσμα και έναν πραγματικό αριθμό. Το διάνυσμα παριστάνει τον άξονα πάνω στον οποίο θα γίνει η περιστροφή και ο πραγματικός αριθμός παριστάνει το κατά πόσο θα γίνει η περιστροφή δεξιόστροφα.

Ένας άλλος τρόπος αναπαράστασης περιστροφής είναι τα διανύσματα του Euler.Πρόκειται για ένα τρισδιάστατο διάνυσμα το οποίο είναι της μορφής <x,y,z> όπου x είναι η περιστροφή που πρέπει να υποστεί το αντικείμενο σε ακτίνια πάνω στον άξονα των x, y αντιστοίχως για τον άξονα των y και ομοίως z για τον άξονα των z. Οι περιστροφές εκτελούνται πρώτα στον άξονα z μετά στον y και τέλος στον x στις περιστροφές κατά Euler.Όμως λόγω του ότι τα ακτίνια είναι λιγότερο διαδεδομένα η LSL παρέχει την δυνατότητα να μετατρέψουμε μοίρες σε ακτίνια μέσω της συνάρτησης DEG_TO_RAD…

```
<30.0 , 23.0 , 136.0>*DEG_TO_RAD
```

Η παραπάνω σειρά κώδικα ουσιαστικά μετατρέπει το διάνυσμα από μοίρες σε ακτίνια. Επίσης υπάρχουν οι σταθερές PI_BY_TWO(90 μοίρες), PI(180 μοίρες) και TWO_PI(360 μοίρες) για συγκεκριμένες γωνίες.

Για μετατροπή μεταξύ του ενός ή του άλλου τρόπου περιστροφής η LSL παρέχει τις μεθόδους llEuler2Rot<vector vecInRadians> και llRot2Euler<rotation rot>. Η πρώτη παίρνει σαν όρισμα ένα διάνυσμα Euler και το μετατρέπει σε LSL quaternion και η δεύτερη παίρνει σαν όρισμα μια μεταβλητή τύπου rotation(LSL quaternion) και το μετατρέπει στο αντίστοιχο διάνυσμα Euler.

Το ακόλουθο script περιστρέφει τον άξονα ενός φύλου μιας πόρτας.

Σημείωση: Η σύνθεση δύο γωνιών συμβολίζεται με το σύμβολο του πολλαπλασιασμού * .

```
DoorMechanism(integer angleToRot)       //Η μέθοδος περιστροφής
{
   rotation rot = llGetRot();          //αρχικά παίρνουμε την global περιστροφή
   vector angle = llRot2Euler(rot);       //και την μετατρέπουμε σε διάνυσμα Euler
   angle.z += angleToRot*DEG_TO_RAD;   //προσθέτουμε την γωνία angleToRot
   rot = llEuler2Rot(angle);          //και μετατρέπουμε πάλι σε rotation
   llSetRot(rot);          // το οποίο χρησιμοποιούμε για περιστροφή
}


default
{
  state_entry()
   {
     llListen(93,"frame1","","open");
   }
   listen(integer channel, string name, key id, string message)
   {
      state open;
   }

}

state open
{
   state_entry()
   {
      DoorMechanism(90);
      llSetTimerEvent(3.0);
   }
      timer()
   {
      DoorMechanism(-90);
      llSetTimerEvent(0);
      state default;
   }
}
```

Πρόκειται για ένα σύνθετο (linked) αντικείμενο που αποτελείται από ένα ορθογώνιο παραλληλεπίπεδο και έναν πολύ λεπτό κύλινδρο ο οποίος αποτελεί τη ρίζα του σύνθετου αντικειμένου και στον οποίο είναι εφαρμοσμένος ο κώδικας.

Αρχικά ορίζουμε την μέθοδο η οποία με όρισμα έναν ακέραιο αριθμό που αντιπροσωπεύει την γωνία που θέλουμε να περιστρέψουμε το σύνθετο αντικείμενο προκαλεί την περιστροφή ως προς την τρέχουσα θέση και περιστροφή του αντικειμένου. Αναλυτικότερα, ορίζουμε μια περιστροφή rot στην οποία αναθέτουμε την τρέχουσα περιστροφή του αντικειμένου μέσω της llGetRot(). Αμέσως μετά ορίζουμε μια περιστροφή Euler με όνομα angle στην οποία αναθέτουμε το διάνυσμα

που προκύπτει από την μετατροπή της περιστροφής rot σε Euler διάνυσμα. Η περιστροφή η οποία θέλουμε να γίνει είναι στον άξονα των z οπότε ορίζουμε , όπως στα πεδία ενός αντικειμένου μιας κλάσης , ότι το μέρος του διανύσματος που περιλαμβάνει τον άξονα z θα πάρει την τιμή του αθροίσματος της τρέχουσας τιμής του συν της γωνίας angleToRot αφού μετατρέψουμε την τελευταία σε radians από μοίρες. Μετά επαναμετατρέπουμε την Euler περιστροφή σε rotation τύπο και την αναθέτουμε στην μεταβλητή rot. Τέλος χρησιμοποιείται η llSetRot με όρισμα την τελική τιμή της rot για να περιστρέψουμε το αντικείμενο.

Το πρόγραμμα έχει δύο καταστάσεις. Την default που αποτελεί τον δέκτη των εισόδων και την open που εκτελεί μια φορά την DoorMechanism με σκοπό να ανοίξει και να κλείσει η πόρτα.

Αρχικά στην default κατάσταση το σύστημα αναμένει για είσοδο στο κανάλι 93 από το αντικείμενο με όνομα frame1 το μήνυμα open το οποίο προκαλεί μετάβαση στην κατάσταση open. Για την κατανόηση των όσων γίνονται το αντικείμενο με όνομα frame1 όταν το αγγίζουμε στέλνει μήνυμα στο κανάλι 93 την λέξη open. Ο λόγος ύπαρξης της κατάστασης open είναι ότι μερικές φορές αν πολλοί χρήστες αγγίξουν το αντικείμενο frame1 τότε πολλαπλά μηνύματα θα δημιουργηθούν και καθώς ο χρόνος απόκρισης ορισμένων συναρτήσεων της LSL είναι σχετικά μεγάλος θα υπήρχε περίπτωση να εκτελεστεί περιστροφή δύο φορές οπότε δεν θα είχαμε το επιθυμητό αποτέλεσμα.

Η κατάσταση open προκαλεί την περιστροφή για το άνοιγμα και εκκινεί ένα timer συμβάν όπου μετά από τρία δευτερόλεπτα η πόρτα περιστρέφεται αντίθετα για να κλείσει. Εκεί τερματίζει το συμβάν timer και επιστρέφει στην κατάσταση default.

Επίσης υπάρχει η περίπτωση ο σχεδιαστής να θέλει να φτιάξει ένα αντικείμενο που θα περιστρέφεται γύρω από κάποιον άξονα πάντα χωρίς να σταματάει. Σε αυτήν την περίπτωση θα χρησιμοποιηθεί η μέθοδος llTargetOmega(vector axis,float spinratePerSec, float amplification).Η συνάρτηση παίρνει ως ορίσματα το διάνυσμα που αντιπροσωπεύει τον άξονα περιστροφής, την ταχύτητα περιστροφής σε ακτίνια ανά δευτερόλεπτο και μια τιμή ενίσχυσης του φαινομένου.

```
default
{
   state_entry()
   {
      llTargetOmega(<0,0,0.2>,-PI,1.0);
   }
}
```

5.2.4 Η μέθοδος llSetPrimitiveParams

Οι ανωτέρω αναφερθέντες μέθοδοι για την επίδραση στα χαρακτηριστικά ενός αντικειμένου είναι σχεδιασμένες να αποδίδουν καλά όταν καλούνται για μια μεταβολή. Όσο πιο συχνά χρειάζεται να τις καλέσουμε τόσο περισσότερος χρόνος απαιτείται καθώς κάθε κλήση τους θέτει το script μας ανενεργό για 0.2 δευτερόλεπτα. Πέρα από αυτό υπάρχουν πεδία του αντικειμένου τα οποία δεν έχουμε την δυνατότητα με τα παραπάνω να τα επηρεάσουμε όπως ο δείκτης διαφάνειας, το αρχικό σχήμα, η γυαλάδα , το fullbright ή ακόμη και η αλλαγή γεωμετρικού σχήματος ή του αν το αντικείμενο θα θεωρείται φυσικό ή όχι(physics).

Για όλα τα παραπάνω υπάρχει η μέθοδος llSetPrimitiveParams(list rules) η οποία δέχεται σαν όρισμα μια λίστα κανόνων μεταβολών. Εφόσον πρόκειται για λίστα έχουμε την δυνατότητα με μία μόνο κλήση να προκαλέσουμε πολλαπλές μεταβολές σε ένα αντικείμενο. Λόγω πληθώρας πιθανών μεταβολών μέσω της llSetPrimitiveParams, θα αναφερθούν μόνο μερικές από τις βασικές μεταβολές που μπορούν να πραγματοποιηθούν με αυτήν καθώς και πώς να οριστεί ένα γεωμετρικό σχήμα ως πηγή φωτός. Περισσότερες πληροφορίες σχετικά με αυτήν την μέθοδο μπορούν να βρεθούν στον παρακάτω σύνδεσμο…
http://wiki.secondlife.com/wiki/LlSetPrimitiveParams .

Αλλαγή γεωμετρικού σχήματος

Το ακόλουθο script αλλάζει το σχήμα του αντικειμένου σε κύλινδρο όταν κάποιος χρήστης το αγγίξει και το επαναφέρει σε ορθογώνιο παραλληλεπίπεδο μετά από πέντε δευτερόλεπτα.

```
default
{
  state_entry()
  {
    llSetPrimitiveParams([PRIM_TYPE,PRIM_TYPE_CYLINDER ,
                PRIM_HOLE_DEFAULT,
                <0.00, 1.0, 0.0>,   // cut
                0.0,            // hollow
                <0.0, 0.0, 0.0>,    // twist
                <1.0, 1.0, 0.0>,    // top_size
                <0.0, 0.0, 0.0>     // top_Shear
                ]);
  }
```

```
    touch_start(integer num)
    {
      llSetPrimitiveParams([PRIM_TYPE,PRIM_TYPE_BOX ,
                  PRIM_HOLE_DEFAULT,
                  <0.00, 1.0, 0.0>,  // cut
                  0.0,           // hollow
                  <0.0, 0.0, 0.0>,   // twist
                  <1.0, 1.0, 0.0>,   // top_size
                  <0.0, 0.0, 0.0>    // top_Shear
                  ]);
      llSetTimerEvent(5.0);
    }

    timer()
    {
      llSetPrimitiveParams([PRIM_TYPE,PRIM_TYPE_CYLINDER ,
                  PRIM_HOLE_DEFAULT,
                  <0.00, 1.0, 0.0>,  // cut
                  0.0,           // hollow
                  <0.0, 0.0, 0.0>,   // twist
                  <1.0, 1.0, 0.0>,   // top_size
                  <0.0, 0.0, 0.0>    // top_Shear
                  ]);
       llSetTimerEvent(0);
    }
  }
```

Όπως βλέπουμε η σύνταξη σε αυτήν την περίπτωση για τον κύλινδρο είναι της μορφής …

```
llSetPrimitiveParams ([
                        PRIM_TYPE,
                        PRIM_TYPE_CYLINDER,
                        integer hole_shape,
                        vector cut,
                        float hollow,
                        vector twist,
                        vector top_size,
                        vector top_shear
                    ]) ;
```

όπου για integer hole_shape χρησιμοποιείται η σταθερά PRIM_HOLE_DEFAULT δηλαδή το προκαθορισμένο σχήμα. Επίσης παρατηρούμε ότι όλα τα στοιχεία τις λίστας περικλείονται από αγκύλες πέρα από παρενθέσεις. Τα στοιχεία και οι μεταβολές χωρίζονται με κόμμα οπότε κατά την προσθήκη επιπλέον μεταβολών αρκεί να προστεθεί το κόμμα μετά από το top_shear και να ξεκινήσουμε με την νέα μεταβολή κάποιου χαρακτηριστικού του αντικειμένου μας.

Αλλαγή χρώματος και διαφάνειας και προσθήκη texture

Πέρα από την llSetColor(vector color, integer face) και αντίστοιχα την llSetAlpha(float alpha, integer face) μπορούν να πραγματοποιηθούν η αλλαγή χρώματος και η αλλαγή τιμής διαφάνειας με την llSetPrimitiveParams. Επίσης είναι δυνατόν να αλλάξει η υφή του αντικειμένου(texture) Στο πρόγραμμα που ακολουθεί φαίνεται πώς μπορεί να γίνουν αυτά με μία μόνο κλήση της μεθόδου.

```
default
{
  state_entry()
  {
    llSetPrimitiveParams([
                PRIM_COLOR,              //ορίζει το χρώμα του αντικειμένου
                ALL_SIDES,               //σε όλες τις πλευρές
                <1.0,1.0,1.0>,           //διάνυσμα χρώματος
                1.0,                     //διαφάνεια, τιμές από 0.0 έως 1.0
                PRIM_TEXTURE,            //ορίζει την υφή (texture)
                ALL_SIDES,               //σε όλες τισ πλευρές
                "metalfloor",            //σε metalfloor που βρίσκεται στα contents
                <1.0,1.0,0.0>,           //επαναλήψεις ως προς μήκος και ύψος
                <0.0,0.0,0.0>,           //μετατοπισμένο από το΄κέντρο
                0.0 //περιστραμένο κατά 0.0 ακτίνια ως προς z'z
                ]);
  }
  touch_start(integer num)
  {
    llSetPrimitiveParams([
                PRIM_COLOR,              //αλλαγή χρώματος σε σκούρο γκρί
                ALL_SIDES,               //και θέτουμε το alpha σε τιμή 0.2
                <0.2,0.2,0.2>,           //ώστε το αντικείμενο να γίνει διάφανο
                0.2,
                PRIM_TEXTURE,            //αλλαγή σε BLANK texture
                ALL_SIDES,
                TEXTURE_BLANK,
                <1.0,1.0,0.0>,
                <0.0,0.0,0.0>,
                0.0
                ]);
    llSetTimerEvent(5.0);
  }




  timer()
  {
```

```
        llSetPrimitiveParams([
                PRIM_COLOR,                 //επιστροφή στην αρχική κατάσταση
                ALL_SIDES,
                <1.0,1.0,1.0>,
                1.0,
                PRIM_TEXTURE,
                ALL_SIDES,
                "metalfloor",
                <1.0,1.0,0.0>,
                <0.0,0.0,0.0>,
                0.0
                ]);
        llSetTimerEvent(0);
    }
}
```

Η σύνταξη για την αλλαγή χρώματος και την αλλαγή διαφάνειας είναι llSetPrimitiveParams([PRIM_COLOR, integerhttp://wiki.secondlife.com/wiki/Integer face, vector color, float alpha]);

ενώ για την αλλαγή υφής είναι…
llSetPrimitiveParams([PRIM_TEXTURE,integer face,string texture, vector repeats, vector offsets, float rotation_in_radians]);

Αλλαγή της τιμής FULLBRIGHT και δημιουργία φωτεινής πηγής.

Η επιλογή FULLBRIGHT που υπάρχει στον viewer αυξάνει την φωτεινότητα στο 100% σε όλες τις επιφάνειες ενός αντικειμένου κάνοντάς το να φαίνεται σαν να φωτίζεται ταυτόχρονα σε όλες τις πλευρές. Όμως αυτό δεν μετατρέπει το αντικείμενο σε φωτεινή πηγή. Μέσω της llSetPrimitiveParams είναι δυνατόν να οριστεί ένα αντικείμενο ως φωτεινή πηγή συγκεκριμένου χρώματος και συγκεκριμένης ακτίνας φωτισμού.

Η LSL στηρίζεται σε OpenGL και κατά συνέπεια έχει τους ίδιους περιορισμούς. Ο μέγιστος αριθμός φωτεινών πηγών είναι οκτώ εκ των οποίων δύο είναι δεσμευμένες θέσεις για τον ήλιο και το φεγγάρι. Αυτό κάνει τον πιο αποτελεσματικό τρόπο φωτισμού ενός χώρου να είναι η εφαρμογή FULLBRIGHT σε όλα τα αντικείμενα του χώρου καθώς επίσης και η προσθήκη γυαλάδας glow. Στο παρακάτω script αλλάζουμε την γυαλάδα και το FULLBRIGHT ενός αντικειμένου και το μετατρέπουμε επίσης σε φωτεινή πηγή.

(Σημείωση : Η Point Light λειτουργία δεν συμπεριφέρεται σωστά σε megaprims. Είναι πιθανόν να μην είναι καν ορατή η να είναι ορατή μόνο σε συγκεκριμένα τμήματα των megaprims)

```
default
{
   touch_start(integer num)
   {
      llSetPrimitiveParams([
                        PRIM_GLOW,                    //ορισμός γυαλάδας
                        ALL_SIDES,                    //σε όλες τις πλευρές
                        1.0,                          //τιμές από 0.0 - 1.0
                    PRIM_FULLBRIGHT,ALL_SIDES,TRUE, //FULLBRIGHT σε όλες τις πλευρές
                        PRIM_POINT_LIGHT,TRUE,     //Δημιουργία φωτεινής πηγής
                        <1.0,1.0,1.0>,          // light color vector range: 0.0-1.0 *3
                        1.0,                    // intensity    (0.0-1.0)
                        20.0,                   // radius       (.1-20.0)
                        0.6                     // falloff      (.01-2.0)
                  ]);
      llSetTimerEvent(5.0);
   }
   timer()
   {
      llSetPrimitiveParams([
                        PRIM_GLOW,                    //ορισμός γυαλάδας
                        ALL_SIDES,                    //σε όλες τις πλευρές
                        0.0,                          //τιμές από 0.0 - 1.0
                        PRIM_FULLBRIGHT,ALL_SIDES,FALSE,//FULLBRIGHT off
                        PRIM_POINT_LIGHT,FALSE, // αφαίρεση φωτεινής πηγής
                        <1.0,1.0,1.0>,          // light color vector range: 0.0-1.0 *3
                        1.0,                    // intensity    (0.0-1.0)
                        10.0,                   // radius       (.1-20.0)
                        0.6                     // falloff      (.01-2.0)
                  ]);
      llSetTimerEvent(0);
   }
}
```

Όταν αγγίζουμε το αντικείμενο το αντικείμενο αποκτά γυαλάδα και το FULLBRIGHT ενεργοποιείται ενώ ταυτόχρονα το αντικείμενο γίνεται φωτεινή πηγή. Επίσης ένα timer συμβάν ενεργοποιείται και μετά από πέντε δευτερόλεπτα το αντικείμενο μας επιστρέφει στην αρχική του κατάσταση. Η σύνταξη της μεταβολής των παραμέτρων του αντικειμένου είναι αντίστοιχα…

Glow:

llSetPrimitiveParams ([PRIM_GLOW, integer face, float intensity]);

FULLBRIGHT:

llSetPrimitiveParams([PRIM_FULLBRIGHT, integer face, integer boolean]);

Point Light:

llSetPrimitiveParams([PRIM_POINT_LIGHT,integer boolean,vector color,float intensity,float radius,float falloff]);

5.3 Άλλες χρήσιμες μέθοδοι

Σε αυτήν την μελέτη θα χρησιμοποιηθούν επίσης δύο μέθοδοι για να έχουμε την δυνατότητα να γνωρίζουμε ποια ενσάρκωση άγγιξε κάποιο αντικείμενο και την τρέχουσα θέση ταχύτητα και περιστροφή της συγκεκριμένης ενσάρκωσης. Οι μέθοδοι αυτές είναι η llDetectedKey(integer index) και η llGetObjectDetails(list features). Η πρώτη επιστρέφει το key id οποιουδήποτε αλληλεπιδρά με κάποιο detection συμβάν. Τέτοια συμβάντα είναι τα collision, collision_start, collision_end, sensor, touch, touch_start, touch_end. Το integer index παίρνει τιμές από 0 έως num_detected μείον. Η δεύτερη μπορεί να επιστρέψει στοιχεία όπως όνομα, περιγραφή, ταχύτητα, περιστροφή, τρέχουσα θέση και άλλα. Επίσης στο script που ακολουθεί χρησιμοποιείται και η llList2String(list source, integer index) που μετατρέπει τα στοιχεία μιας λίστας σε τύπο string κάνοντας typecast.Τέλος υπάρχει και η μέθοδος llFrand(float max) που επιστρέφει μια τυχαία τιμή από 0 έως max.

```
    key uuid;                           //Ορίζουμε την global μεταβλητή uuid τύπου key

    default
    {
      touch_start(integer num)
      {
        uuid=llDetectedKey(0);          //Η uuid παίρνει την τιμή του key id της
                                        //ενσάρκωσης που αγγίζει το αντικείμενο
        state info;
      }
    }
state info
    {
      state_entry()
      {
       float n=llFrand(2.0)+1;          //Τυχαία τιμή από 0-2.0 +1 δηλαδή 1.0-3.0
       llSetTimerEvent(n);
      }
      timer()
    {
       list a=llGetObjectDetails(uuid, ([       //αποθηκεύουμε σε μια list τα στοιχεία που
                                                //φαίνονται παρακάτω
                                       OBJECT_NAME,         //όνομα
                                       OBJECT_POS,          //θέση
                                       OBJECT_ROT,          //περιστροφή
                                       OBJECT_VELOCITY]));  //ταχύτητα
        llShout(0,"\nUUID: "      + (string)llDetectedKey(0) +  //μετατροπή σε string
              "\nName: \""      + llList2String(a,0) + "\"" +  //και παρουσίαση
              "\nPosition: "    + llList2String(a,1) + "\"" +
```

```
            "\nRotation: "    + llList2String(a,2) + "\"" +
            "\nVelocity: "    + llList2String(a,3));
    }
    touch_start(integer num)
    {
        llSetTimerEvent(0);
        state default;
    }
}
```

Το παραπάνω τμήμα κώδικα κάνει το αντικείμενο στο οποίο ανήκει να αποθηκεύσει αρχικά το key uuid οποιουδήποτε το αγγίξει και μετά συλλέγει κάθε n (n τυχαία τιμή 1.0-3.0) δευτερόλεπτα τα στοιχεία: όνομα, θέση, περιστροφή και τρέχουσα ταχύτητα και με shout στο κανάλι 0 (δηλαδή το chat channel) τα εμφανίζει. Μόλις ο χρήστης ξαναγγίξει το αντικείμενο αυτή η διαδικασία σταματάει. Αναλυτικότερες πληροφορίες σχετικά με την llGetObjectDetails και τις παραμέτρους της, μπορούν να βρεθούν στο http://wiki.secondlife.com/wiki/LlGetObjectDetails.

Εμφάνιση ονόματος, θέσης, περιστροφής και ταχύτητας του χρήστη που άγγιξε το αντικείμενο.

5.4 LSL Physics και μέθοδοι χρήσης

Η LSL παρέχει ένα ήδη έτοιμο σύστημα κανόνων φυσικής οι οποίοι μπορούν να επιδράσουν σε ένα αντικείμενο το οποίο είναι επιλεγμένο να είναι physical. Για να σημειωθεί ένα αντικείμενο ως physical αρκεί να το ορίσουμε στις επιλογές του αντικειμένου. Αν ορίσουμε την θέση ενός αντικειμένου ψηλότερα από το έδαφος για παράδειγμα, όταν το μετατρέψουμε σε physical το πρώτο πράγμα που θα παρατηρήσει κάποιος είναι ότι το αντικείμενο θα πέσει στο έδαφος.

Ωστόσο υπάρχουν περιπτώσεις που θέλουμε το αντικείμενό μας να αλλάξει κατάσταση κατά την εκτέλεση κάποιου script. Σε αυτήν την περίπτωση μπορεί να χρησιμοποιηθεί η μέθοδος llSetPrimitiveParams με list rules το [PRIM_PHYSICS, integer Boolean]. Επίσης με την ίδια μέθοδο μπορούμε να ορίσουμε τύπο σχήματος ως προς το πώς θα εφαρμόζονται πάνω στο αντικείμενο οι φυσικοί κανόνες καθώς και υλικό του αντικειμένου στο οποίο μπορεί να οριστεί συντελεστής τριβής , πυκνότητα , συντελεστής βαρύτητας και ελαστικότητα.

```
default
{
  touch_start(integer num)
  {
    llSetPrimitiveParams([                    //physics=true
                PRIM_PHYSICS,
                TRUE,
                PRIM_MATERIAL,                //και ότι το υλικό είναι λάστιχο
                PRIM_MATERIAL_RUBBER

            ]);
    llSetTimerEvent(5.0);
  }

  timer()
  {
    llSetPrimitiveParams([                    //physics=false
                PRIM_PHYSICS,
                FALSE,
                PRIM_POSITION,                //και μεταφορά στην θέση
                <143.,137.,37.>               //        vector pos
            ]);
     llSetTimerEvent(0);
  }
}
```

Η μέθοδος llSetForce και llApplyImpulse

Η μηχανή φυσικής που έχει ενσωματωμένη το Second Life επιτρέπει στον σχεδιαστή μέσα από την γλώσσα LSL να ασκήσει δύναμη σε κάποιο physical αντικείμενο. Το αντικείμενο ως αναμενόμενο έχει κάποιον συντελεστή τριβής και πιθανώς κάποια δύναμη αντίδρασης. Αν η δύναμη που θα ασκηθεί είναι μεγαλύτερη από την αντίδραση τότε θα αρχίσει να επιταχύνει κατά την φορά της δύναμης αναλόγως με το πόσο μεγάλη είναι η διαφορά μεταξύ της ασκούμενης δύναμης και της συνιστώσας των δυνάμεων αντίδρασης. Έτσι , είναι εύκολο να φανταστεί κανείς ότι ένα κυβικό σχήμα θα μετακινηθεί στο οριζόντιο επίπεδο με μεγαλύτερη δυσκολία από ότι ένα σφαιρικό ενώ και τα δύο θα έχουν την ίδια δυσκολία να κινηθούν κατακόρυφα αν η μάζα τους είναι η ίδια. Για να ασκηθεί μία δύναμη σε κάποιο αντικείμενο πρέπει να χρησιμοποιηθεί η ακόλουθη σύνταξη…

llSetForce(vector force, integer local);

Η μέθοδος llSetForce παίρνει σαν όρισμα ένα διάνυσμα που αντιπροσωπεύει τόσο την κατεύθυνση όσο και την ένταση της δύναμης που ασκείται. Επίσης το δεύτερο όρισμα της μεθόδου είναι η τιμή TRUE ή FALSE και αντιπροσωπεύει το αν το σύστημα θα χρησιμοποιήσει τοπικές ή σφαιρικές συντεταγμένες.

Επίσης μέσω της llApplyImpulse(vector force , integer local) είναι δυνατόν να ασκήσουμε στιγμιαία δύναμη σε κάποιο αντικείμενο. Παραδείγματος χάρη ένας άνθρωπος κλωτσάει μια μπάλα. Στιγμιαία ασκεί δύναμη στην μπάλα προς κάποια κατεύθυνση και αυτή εκτινάσσεται προς την επιθυμητή κατεύθυνση. Έτσι και εδώ αν ασκηθεί στιγμιαία μια κατακόρυφη δύναμη σε ένα αντικείμενο , το αντικείμενο θα εκτιναχτεί ψηλά αλλά μετά θα ξαναπέσει στο έδαφος. Η llApplyImpulse αγνοεί την τρέχουσα αντίδραση εκτός και πρόκειται για κάποια αντίδραση λόγω σύγκρουσης.

```
float vel;

default
{
   touch_start(integer num)
   {
      llSetPrimitiveParams([                                  //physics=true
                        PRIM_PHYSICS,
                        TRUE,
                        PRIM_MATERIAL,                  //και ότι το υλικό είναι λάστιχο
                        PRIM_MATERIAL_RUBBER
                              ]);
```

```
      llSetForce(<-2.0,-0.15,0.0>,FALSE);                //ορισμός δύναμης
      llSetTimerEvent(0.5);                              //και timer συμβάντος
   }
   timer()
   {
      vel=llVecMag(llGetVel());        //το vel παίρνει την τιμή του μέτρου
                                       //της τρέχουσας ταχύτητας
      if(vel>8)
      {
      llSetTimerEvent(0);
      llSetForce(<0.1,0.0,0.0>,FALSE);
      llApplyImpulse(<0.0,0.0,4.0>,FALSE);
      state cooldown;
      }
   }
}
state cooldown
{
   state_entry()
   {
      llSetTimerEvent(0.5);
   }
   timer()
   {
      vel=llVecMag(llGetVel());
      if(vel<0.5)
      {
         llSetPrimitiveParams([
                        PRIM_PHYSICS,
                        FALSE
                     ]);
         vector x=<0.0,0.0,0.0>;
         while(x!=<143.0,137.0,32.0>)
         {
            llSetPrimitiveParams([
                        PRIM_POSITION,      //και μεταφορά στην θέση
                        <143.,137.,32.>     //      143.0,137.0,32.0
                     ]);
            x=llGetPos();
         }
         state default;
      }
   }
}
```

Αρχικά το αντικείμενο βρίσκεται κατά προτίμηση σε κάποιο ύψος από το έδαφος. Η physics κατάστασή του ορίζεται αληθής οπότε το αντικείμενο αρχίζει να πέφτει. Επίσης το υλικό του αντικειμένου ορίζεται στο προκαθορισμένο για τον LSL compiler, rubber, του ασκείται σταθερή δύναμη με διάνυσμα δυνάμεως το <-2.0,-0.15,0.0> σε σφαιρικές συντεταγμένες και ένα timer συμβάν ενεργοποιείται κάθε 0.5 δευτερόλεπτα. Το αντικείμενο αρχίζει να επιταχύνει προς την διεύθυνση της δυνάμεως.

Το timer συμβάν ουσιαστικά κάνει έλεγχο του μέτρου της ταχύτητας του αντικειμένου. Για να γίνει αυτό χρησιμοποιείται η μέθοδος llVecMag(vector x); όπου επιστρέφει το μέτρο του διανύσματος που παίρνει σαν όρισμα και επίσης χρησιμοποιείται και η llGetVel(); η οποία επιστρέφει την τρέχουσα ταχύτητα του αντικειμένου σε μορφή τρισδιάστατου διανύσματος. Επίσης ορίζεται μια συνθήκη κατά την οποία όταν το μέτρο της ταχύτητας είναι μεγαλύτερο από οχτώ τότε το timer συμβάν ακυρώνεται, μια νέα δύναμη εφαρμόζεται (αντίθετη και πολύ μικρότερου μέτρου από την προηγούμενη) με σκοπό να επιβραδύνει το αντικείμενο ενώ επίσης ασκείται και μία κατακόρυφη δύναμη στιγμιαία με την μέθοδο llApplyImpulse που εκτοξεύει το αντικείμενο ψηλά και αυτό επιστρέφει στο έδαφος. Η κατάσταση του αντικειμένου πλέον ορίζεται ως cooldown.

Στην κατάσταση cooldown αρχικά ξεκινάει νέο timer συμβάν το οποίο κάθε 0.5 δευτερόλεπτα κάνει έλεγχο για το μέτρο της ταχύτητας. Αν το μέτρο της ταχύτητας είναι μικρότερο του 0.5 τότε το αντικείμενο σταματάει να θεωρείται physical. Επίσης ορίζεται και αρχικοποιείται ένα διάνυσμα x το οποίο συγκρίνεται σε έναν βρόγχο while με κάποιο συγκεκριμένο διάνυσμα αρχικής θέσης του αντικειμένου. Όσο τα δύο διανύσματα διαφέρουν το αντικείμενο καλείται να μεταφερθεί στις συντεταγμένες που έχουν οριστεί ως συντεταγμένες αρχικής θέσης (διάνυσμα σύγκρισης) και το διάνυσμα x παίρνει την τιμή της τρέχουσας θέσης του αντικειμένου μας. Όταν τα δύο διανύσματα γίνουν ίσα τότε η κατάσταση του αντικειμένου επιστρέφει σε default. Ο λόγος ύπαρξης του βρόγχου while είναι ότι κάθε μετατόπιση ενός αντικειμένου δεν μπορεί να ξεπεράσει τα 10 μέτρα ως προς την σχετική του θέση.

Η μέθοδος llMoveToTarget

Σε κάποιο αντικείμενο physical είναι δυνατόν να οριστεί κάποια κίνηση προς μια συγκεκριμένη θέση. Αυτό επιτυγχάνεται με την μέθοδο llMoveToTarget(vector targetloc, float time). Η μέθοδος δέχεται σαν ορίσματα ένα διάνυσμα που αντιπροσωπεύει την θέση στην οποία θέλουμε το αντικείμενο να κινηθεί καθώς επίσης και έναν αριθμό κινητής υποδιαστολής ο οποίος αντιπροσωπεύει το χρονικό διάστημα που θα χρειαστεί ώστε το αντικείμενό μας να φτάσει στον προορισμό του. Επίσης ένα ακόμη χαρακτηριστικό αυτής της μεθόδου είναι ότι προκαλεί την κίνηση ακόμη και σε ενσαρκώσεις(avatars) αν είναι σε αυτούς προσκολλημένο(attached) κάποιο αντικείμενο που την περιέχει.

5.5 Δυναμική προσθήκη αντικειμένων στο χώρο.

Η μέθοδος llRezObject(string objectname, vector position, vector velocity, rotation itemrot, integer param) παρέχει την δυνατότητα να δημιουργήσουμε ένα αντικείμενο που υπάρχει ήδη στα περιεχόμενα του αντικείμενου με το script σε συγκεκριμένη θέση position , με περιστροφή itemrot και αν το αντικείμενο είναι physical οποιαδήποτε τιμή δώσουμε στο velocity.Επίσης το integer param είναι μια τιμή που το αντικείμενο παίρνει κατά την δημιουργία του με σκοπό να μπορούν να εφαρμοστούν συμβάντα on_rez τα οποία εκτελούνται αντί του state_entry για αντικείμενα που έχουν γίνει rez στο χώρο. Κατά προτίμηση, αντικείμενα τα οποία δεν είναι απαραίτητο να μένουν στο χώρο, θα ήταν προτιμότερο να τα ορίσουμε ως temporary έτσι ώστε μετά από ένα λεπτό να εξαφανιστούν και να μην φορτώσουν τον χώρο της με περιορισμένης χρήσης αντικείμενα.

Ακολουθεί ένα script της επίδειξη των δυνατοτήτων.

```
Default
{
   integer multiplier=5;                                  //πολλαπλαστιαστής ταχύτητας

   state_entry()
   {
      vector startvec=<0.0,330.0,330.0>*DEG_TO_RAD;  //αρχική περιστροφή σε μοίρες
                                                                  //και μετατροπή σε ακτίνια
     rotation startrot=llEuler2Rot(startvec);          //μετατροπή από euler σε rotation
     llSetPrimitiveParams([                                  //εφαρμογή της περιστροφής
                    PRIM_ROTATION,
                    startrot
                  ]);
```

```
    }

    touch(integer num)                          //όσο διαρκεί το touch event
    {                                           //κάθε 0.2 δευτερόλεπτα
      llSetTimerEvent(0.5);                     //εκτελείται το συμβάν timer
    }

    touch_end(integer num)                      //στο τέλος του touch
    {
      llSetTimerEvent(0.0);                     //το timer event ακυρώνεται
      vector offset=<1.8,0.0,0.0>;              //ορισμός της απόκλισης σε σχέση με
                                                //μηδενική περιστροφή
      llRezObject(                              //δημιουργία αντικειμένου
      «bullet»,                                 //με όνομα bullet
      llGetPos()+offset*llGetRot(),             //νέα θέση = τρέχουσα συν το offset
                                                //περιστραμένο κατά τρέχουσα περιστροφή
      llVecNorm(offset*llGetRot())*multiplier,      //ταχύτητα = διεύθυνση offset επί
                                                //πολλαπλασιαστής
      <0.0,0.0,0.0,1.0>,                        //μηδενικό rotation
                                                 //δείκτης 1
      );
      Multiplier=5;                             //επανάθεση του 5 στον πολλαπλασιαστή
    }
    timer()
    {
      multiplier+=3;                             //όποτε εκτελείται το timer event
    }                                           //ο πολλαπλασιαστής αυξάνει κατά τρία

}
```

Το παραπάνω τμήμα κώδικα δημιουργεί ένα αντίγραφο κάποιου αντικειμένου σε κάποια απόσταση από το αντικείμενο που περιέχει το script. Η απόσταση αυτή (offset) έχει οριστεί σε σχέση με το αντικείμενο που περιέχει το script όταν το δεύτερο βρίσκεται σε μηδενική περιστροφή γύρω από όλους άξονες.

Αναλυτικότερα , αρχικά ορίζεται στην κατάσταση default της ακέραιος αριθμός που αποτελεί τον πολλαπλασιαστή αρχικής ταχύτητας. Αμέσως μετά ορίζονται οι συνθήκες της οποίες θα βρεθεί το αντικείμενο με το που μπαίνει στην κατάσταση default. Εδώ ορίζεται ένα διάνυσμα περιστροφής του Euler στο οποίο ανατίθεται η περιστροφή κατά <0.0,330.0,330,0> αφού πρώτα μετατραπεί σε ακτίνια από μοίρες με την DEG_TO_RAD. Η περιστροφή εκτελείται και έτσι έχουμε της συνθήκες εισόδου στην κατάσταση default.

Της ορίζεται ένα συμβάν touch στο οποίο όσο ο χρήστης αγγίζει το αντικείμενο κάθε 0.5 δευτερόλεπτα εκτελεί ένα timer συμβάν το οποίο με την σειρά του αυξάνει τον πολλαπλασιαστή ταχύτητας κατά τρία. Μόλις ο χρήστης αφήσει το αντικείμενο το συμβάν touch_end εκκινείται το

οποίο αρχικά ακυρώνει το χρονόμετρο του timer συμβάντος και το απενεργοποιεί. Εκεί ορίζεται πλέον η απόκλιση του αντικειμένου που θα δημιουργηθεί από το αντικείμενο που περιέχει τον παραπάνω κώδικα με βάση το πού θα βρισκόταν το πρώτο σε σχέση με το δεύτερο σε μηδενική περιστροφή σε όλους της άξονες.

Με χρήση της llRezObject δημιουργούμε ένα αντικείμενο με βάση το αντικείμενο που έχουμε στα περιεχόμενα με όνομα bullet στην θέση που αντιστοιχεί στην τρέχουσα θέση του αντικειμένου που περιέχει το script συν το offset μετά από περιστροφή ίση με την τρέχουσα περιστροφή του αντικειμένου ρίζα. Ακολουθεί η ταχύτητα η οποία προκύπτει από το γινόμενο του πολλαπλασιαστή ταχύτητας και της κανονικοποίησης του διανύσματος που αντιστοιχεί στην περιστραμμένη απόκλιση. Το αντικείμενο που δημιουργείται έχει μηδενική τοπική περιστροφή και του αποδίδεται ο δείκτης ένα. Αμέσως μετά ο πολλαπλασιαστής ταχύτητας επαναφέρεται στην αρχική του τιμή.

Το παραπάνω τμήμα κώδικα μπορεί να εφαρμοστεί σε ένα κανόνι παραδείγματος χάρη το οποίο εκτοξεύει μικρές μπάλες της την διεύθυνση του και όσο κάποιος συνεχίζει να αγγίζει το κανόνι τόσο αυξάνει την αρχική ταχύτητα της βολής.

6. OSSL και δημιουργία Non Player Characters (NPCs)

Όλες οι προηγούμενες μέθοδοι βοηθούν στην δημιουργία της αίσθησης του ρεαλισμού. Παρόλα αυτά ο κόσμος μας θα φαίνεται πάντα άδειος αν δεν υπάρχει κάποια αλληλεπίδραση με πράκτορες ή άλλους χρήστες. Για την δεύτερη κατηγορία δεν μπορεί να γίνει τίποτε παραπάνω σε επίπεδο λογισμικού. Ωστόσο , πράκτορες μπορούμε να δημιουργήσουμε με κώδικα σε C# μέσω κάποιας βιβλιοθήκης όπως η libopenmetaverse ή με μεθόδους της επέκτασης της γλώσσας LSL που χρησιμοποιεί ο OpenSimulator γνωστής και ως OSSL. Δυστυχώς η πρώτη μέθοδος δεν υποστηρίζεται πλήρως πλέον από τον OpenSimulator και έτσι δεν είναι εφικτό στην τρέχουσα έκδοση να προκαλέσουμε κίνηση όπως εμείς επιθυμούμε σε κάποια ενσάρκωση.

6.1 Ρυθμίσεις για να επιτρέπεται η χρήση NPCs στον OpenSimulator.

Για να είναι δυνατή η δημιουργία ενός πράκτορα με τις μεθόδους που μας παρέχονται μέσω OSSL είναι απαραίτητο να γίνουν ορισμένες ρυθμίσεις στο αρχείο Opensim.ini. Μέσα στον φάκελο bin βρίσκεται το αρχείο το οποίο ανοίγει με οποιονδήποτε επεξεργαστή κειμένου. Προτείνεται η χρήση κάποιου επεξεργαστή κειμένου με δυνατότητα μορφοποίησης καθώς η χρήση του σημειωματάριου/Notepad καθιστά δυσκολότερη την ανάγνωση και την τροποποίηση των στοιχείων. Μέσω αναζήτησης βρίσκουμε τα τμήματα [NPC] και [XEngine].

-[NPC] : Σε αυτό το τμήμα αλλάζουμε Enabled σε true

```
[NPC]
    ;# {Enabled} {} {Enable Non Player Character (NPC) facilities}
{true false} false
     Enabled = true
```

-[XEngine] : Σε αυτό το τμήμα επιβεβαιώνουμε ότι το Enabled σε true

```
[XEngine]
    ; Enable this engine in this OpenSim instance
    Enabled = true
```

Επίσης πρέπει να επιτρέπεται η χρήση των osFunctions στο τμήμα XEngine . Αυτό γίνεται με την ρύθμιση

```
;; Allow the use of os* functions (some are dangerous)
     AllowOSFunctions = true
```

…και τέλος ή να οριστεί το OSFunctionThreatLevel σε very high ή να επιτραπεί κάθε μία μέθοδος χωριστά όπου στην δεύτερη περίπτωση η μέθοδοι θα επιτρέπεται να χρησιμοποιηθούν ακόμη και σε περίπτωση που υπερβαίνουν το threat level. Για να γίνει αυτό πρέπει…

```
OSFunctionThreatLevel = VeryHigh
```

ή να χρησιμοποιηθεί η …

`Allow_'OSFunctionName'` και να ακολουθεί η τιμή true η οποιαδήποτε presets απαιτούνται.

Μετά από τις παραπάνω ρυθμίσεις η δημιουργία και η διαχείριση ενός ή περισσοτέρων πρακτόρων είναι πλέον εφικτή.

6.2 Μέθοδοι δημιουργίας και διαγραφής πράκτορα(NPC)

Για την δημιουργία ενός πράκτορα NPC η OSSL μας παρέχει την μέθοδο osNpcCreate(string firstname, string lastname, vector position, string cloneFrom) η οποία επίσης είναι τύπου key. Η μέθοδος παίρνει σαν ορίσματα το first name του NPC όπως το θέλει ο δημιουργός του script , το last name, ένα διάνυσμα για την θέση που θα εμφανιστεί ο πράκτορας καθώς επίσης και το όνομα ενός Notecard το οποίο έχει αποθηκευμένα όλα τα στοιχεία της εμφάνισης του πράκτορα ή το UUID κάποιας ενσάρκωσης η κάποιου πράκτορα που βρίσκεται εκείνη την χρονική στιγμή μέσα στην περιοχή. Η μέθοδος επιστρέφει τιμή τύπου key το οποίο στις περισσότερες περιπτώσεις αποθηκεύεται από τον δημιουργό του κόσμου σε κάποια μεταβλητή αντίστοιχου τύπου καθώς η περεταίρω αλληλεπίδραση με τον πράκτορα απαιτεί το UUID του.

```
default
{
        touch_start(integer num)
        {
                key npc=osNpcCreate(
                                        "Your       own",               //Όνομα
                                         "Doppleganger",                //Επώνυμο
                                        llGetPos()+<1.0,0.0,0.0>,       //τρέχουσα
                                                                        //θέση συν ένα
                                                                        //στον άξονα x'x
                                        llDetectedKey(0)                //key της
                                        );                              //ενσάρκωσης που
                                                                //άγγιξε το αντικείμενο
        }
}
```

Το παραπάνω τμήμα κώδικα δημιουργεί έναν πράκτορα κάθε φορά που κάποιος χρησιμοποιεί/αγγίζει το αντικείμενο που περιέχει το script με όνομα Your own Doppleganger σε απόσταση ένα μέτρο στον άξονα των x'x από το αντικείμενο κατ' εικόνα του οποιουδήποτε χρησιμοποίησε το αντικείμενο.

Στις νεότερες εκδόσεις του OpenSim έχει υπερφορτωθεί η συνάρτηση με το integer options που παρέχει μερικά χαρακτηριστικά στον δημιουργούμενο πράκτορα. Οι επιλογές είναι δύο:

OS_NPC_CREATOR_OWNED

OS_NPC_NOT_OWNED

Η πρώτη επιλογή ορίζει ότι ο πράκτορας ανήκει στον δημιουργό του και μόνο οι εντολές που δίνονται από τον κάτοχο του εκτελούνται ενώ η δεύτερη κάνει τον πράκτορα ελεύθερο προς αλληλεπίδραση με όλους καθώς δεν ανήκει σε κανέναν.

Οι πράκτορες κατά την δημιουργία τους παραμένουν μέσα στον κόσμο ανεξαρτήτως αν ο server απενεργοποιηθεί η όχι. Αποτελούν πλέον μέρος του κόσμου και αποθηκεύονται σαν μέρος του κόσμου. Στην περίπτωση λοιπόν που πρέπει να αφαιρεθεί ένας πράκτορας από την περιοχή χρησιμοποιείται η μέθοδος osNpcRemove(key npc) που παίρνει σαν όρισμα το UUID του πράκτορα που πρέπει να αφαιρεθεί. Η μέθοδος είναι τύπου void και δεν επιστρέφει καμία τιμή.

Αν στο προηγούμενο τμήμα κώδικα προστεθούν οι γραμμές μέσα στο συμβάν touch_start και μετά από την κλήση της μεθόδου osNpcCreate…

```
llSleep(5.0);
osNpcRemove(npc);
```

…τότε το πρόγραμμα παγώνει για 5 δευτερόλεπτα όπως ορίζει η μέθοδος llSleep(float timeInSeconds) και μετά ο πράκτορας που δημιουργήσαμε αφαιρείται από τον κόσμο.

6.3 Μέθοδοι για εύρεση θέσης, περιστροφής ενός πράκτορα και κίνηση.

Για την εύρεση της τρέχουσας θέσης κάποιου πράκτορα γνωρίζοντας το UUID του υπάρχει η μέθοδος osNpcGetPos(key UUID) και η osNpcGetRot(key UUID). Η πρώτη επιστρέφει ένα διάνυσμα/vector που αντιπροσωπεύει την τρέχουσα θέση του πράκτορα ενώ η δεύτερη επιστρέφει μια περιστροφή/rotation. Επίσης υπάρχει η μέθοδος osNpcSetRot(key npc, rotation rot) η οποία ορίζει την περιστροφή του πράκτορα με UUID npc κατά rotation rot. Τέλος η μέθοδος osNpcSay(key npc, string message) κάνει τον πράκτορα να χρησιμοποιήσει το βασικό κανάλι επικοινωνίας και να πει οτιδήποτε περιέχει το message.

Το προηγούμενο πρόγραμμα με προσθήκες ,αρχικά κάνει τον πράκτορα να πει στο βασικό κανάλι επικοινωνίας σε ποια θέση βρίσκεται και τέλος περιστροφή κατά 60 μοίρες.

```
default

{

  touch_start(integer num)
```

```
        {
            key
    npc=osNpcCreate("My","Doppleganger",llGetPos()+<1.0,0.0,0.0>,llDetectedKey(0));
            vector position=osNpcGetPos(npc);
            osNpcSay(npc,(string)position);
            llSleep(5.0);

    osNpcSetRot(npc,osNpcGetRot(npc)+llEuler2Rot(<0.0,0.0,60.0>*DEG_TO_RAD));
            llSleep(5.0);
            osNpcRemove(npc);
        }
}
```

Όπως προαναφέρθηκε αρχικά το αντικείμενο με το συμβάν touch_start δημιουργεί έναν πράκτορα με όνομα My Doppleganger μετατοπισμένο από την αρχική θέση του αντικειμένου που περιέχει το script κατά ένα μέτρο ως προς τον άξονα x'x.

Αμέσως μετά η θέση του πράκτορα βρίσκεται μέσω της μεθόδου osNpcGetPos και αποθηκεύεται σε μία μεταβλητή με όνομα position που είναι τύπου vector. Τότε χρησιμοποιείται η osNpcSay ώστε ο πράκτορας να πεί στο βασικό κανάλι επικοινωνίας την τρέχουσα θέση του η οποία μετατρέπεται σε string τύπο με typecast.

Η λειτουργία του προγράμματος παγώνει για πέντε δευτερόλεπτα και μετά ο πράκτορας περιστρέφεται κατά την θετική φορά κατά εξήντα μοίρες. Η λειτουργία του προγράμματος παγώνει για άλλα πέντε δευτερόλεπτα και τέλος το πράκτορας καταστρέφεται.

Η κίνηση ενός πράκτορα γίνεται με την μέθοδο osNpcMoveToTarget(key npc, vector target, integer options) η οποία δεν επιστρέφει καμία τιμή. Σαν ορίσματα παίρνει το UUID του πράκτορα, ένα διάνυσμα που υποδεικνύει την θέση στην οποία πρέπει να μετακινηθεί ο πράκτορας και τέλος μια τιμή που αντιπροσωπεύει τον τρόπο με τον οποίο ο πράκτορας θα μετακινηθεί στο σημείο π.χ. πετώντας.

Οι πιθανές τιμές του integer options είναι:

-OS_NPC_FLY όπου ορίζει ότι ο πράκτορας πρέπει να πετάξει για να φτάσει στον προορισμό του

-OS_NPC_LAND_AT_TARGET όπου ορίζει ότι ο πράκτορας θα προσγειωθεί μόλις φτάσει στην θέση που πρέπει και που συνδυάζεται με την επιλογή OS_NPC_FLY χρησιμοποιώντας το | μετά από το OS_NPC_FLY.

-OS_NPC_NO_FLY όπου ορίζει ότι ο πράκτορας θα πάει σε κάποια θέση περπατώντας.

Στο προηγούμενο πρόγραμμα προστίθεται κίνηση του πράκτορα κατά 5 μέτρα ως προς x'x μετά επιστροφή στην αρχική θέση και προσγείωση και τέλος περπάτημα κατά 5 μέτρα προς x'x.

```
default
{
   touch_start(integer num)
   {
    key npc=osNpcCreate(
                              "My","Doppleganger",
                              llGetPos()+<1.0,0.0,0.0>,
                              llDetectedKey(0)
                              );
    llSleep(3.0);                       //χρονική καθυστέρηση 3 sec
     osNpcMoveToTarget(                 //μετακίνηση κατά 10 μέτρα πετώντας
                              npc,
                              osNpcGetPos(npc)+<10.0,0.0,0.0>,
                              OS_NPC_FLY
                              );
     llSleep(3.0);                      //χρονική καθυστέρηση 3 sec
     osNpcMoveToTarget(                 //μετακίνηση κατά -10 μέτρα με προσγείωση
                              npc,
                              osNpcGetPos(npc)+<-10.0,0.0,0.0>,
                              OS_NPC_FLY|OS_NPC_LAND_AT_TARGET
                              );
     llSleep(2.0);                      //χρονική καθυστέρηση 2 sec
     osNpcMoveToTarget(                 //μετακίνηση κατά 10 μέτρα περπατώντας
                              npc,
                              osNpcGetPos(npc)+<10.0,0.0,0.0>,
                              OS_NPC_NO_FLY
                              );
     vector position=osNpcGetPos(npc);
     osNpcSay(npc,(string)position);
     llSleep(5.0);                      //χρονική καθυστέρηση 5 sec
     osNpcSetRot(npc,osNpcGetRot(npc)+llEuler2Rot(<0.0,0.0,60.0>*DEG_TO_RAD));
     osNpcSetRot(npc,osNpcGetRot(npc)+llEuler2Rot(<0.0,0.0,60.0>*DEG_TO_RAD));
     llSleep(5.0);                      //χρονική καθυστέρηση 5 sec
     osNpcRemove(npc);
   }
}
```

Τέλος υπάρχει και μία ακόμη μέθοδος να προκαλέσουμε κίνηση σε κάποιον πράκτορα η ενσάρκωση. Προσαρτώντας σε κάποιο μέλος της ενσάρκωσης ένα physical αντικείμενο και με χρήση της llMoveToTarget προκαλείται κίνηση προς το σημείο που ορίζεται στην llMoveToTarget περπατώντας αν το έδαφος είναι λείο και ο πράκτορας είναι προσγειωμένος και πετώντας αν όχι.

6.4 Εμφάνιση πράκτορα και προσαρτημένα αντικείμενα.

Όπως προαναφέρθηκε η εμφάνιση του πράκτορα αποτελεί μία εκ των παραμέτρων της μεθόδου osNpcCreate. Αυτή η παράμετρος μπορεί να πάρει την τιμή είτε κάποιας τρέχουσας ταυτότητας μιας ενσάρκωσης η αντικειμένου η μπορεί να χρησιμοποιηθεί μια κάρτα αποθηκευμένη στον αποθεματικό χώρο του αντικειμένου που περιέχει το script η οποία έχει αποθηκευμένες όλες τις πληροφορίες της εμφάνισης κάποιας ενσάρκωσης ή αντικειμένου. Σε αυτές τις πληροφορίες μέσα μπορεί να αναφέρεται επίσης τι αντικείμενα μεταφέρει ο χαρακτήρας στον αποθεματικό του χώρο καθώς επίσης και πια έχει προσαρτημένα και σε ποιο μέρος. Οι ακόλουθες μέθοδοι δημιουργούν αυτόματα μια κάρτα με συγκεκριμένο όνομα στον αποθεματικό χώρο του αντικειμένου που περιέχει το script.

-osOwnerSaveAppearance(string notecardname);

Η μέθοδος είναι τύπου key και δημιουργεί notecard με όνομα πού παίρνει ως παράμετρο το οποίο περιέχει τα στοιχεία της εμφάνισης του ιδιοκτήτη του αντικειμένου ο οποίος πρέπει να βρίσκεται στην περιοχή (region) την στιγμή της κλήσης της μεθόδου.

-osAgentSaveAppearance(key npc, string notecardname);

Η μέθοδος είναι τύπου key και δημιουργεί notecard με όνομα που παίρνει ως παράμετρο με τα στοιχεία της εμφάνισης της ενσάρκωσης με UUID το περιεχόμενο της παραμέτρου npc. Η ενσάρκωση πρέπει να βρίσκεται στην περιοχή (region) την στιγμή της κλήσης της μεθόδου.

-osNpcSaveAppearance(string notecardname);

Η μέθοδος είναι τύπου key και δημιουργεί notecard με όνομα πού παίρνει ως παράμετρο το οποίο περιέχει τα στοιχεία της εμφάνισης ενός ήδη υπάρχοντος πράκτορα ο οποίος πρέπει να βρίσκεται στην περιοχή (region).

Τέλος η μέθοδος osNpcLoadAppearance(key npc, string notecardname) φορτώνει στον πράκτορα με UUID ίδιο με npc τα στοιχεία που έχουν αποθηκευτεί με κάποια από τις παραπάνω μεθόδους σε notecard με όνομα notecardname.

6.5 Δημιουργία ολοκληρωμένου προγράμματος επίδειξης των ανωτέρω δυνατοτήτων.

```
key npc=NULL_KEY;
default
{
  state_entry()
  {
    llListen(10,"",NULL_KEY,"");                    //listen συμβάν στο κανάλι 10
  }

  listen(integer channel, string name, key id, string msg)
  {
    if (msg != "")                                  //επιλογές για το τι το msg μπορεί να είναι
    {
      if (msg == "create" && npc == NULL_KEY)       // αν msg είναι create και npc =κενή
      {
        osOwnerSaveAppearance("appearance");        //δημιουργία notecard appearance
                                                    //με στοιχεία από τον owner  του αντικειμένου
                                                    //δημιουργία πράκτορα
        npc = osNpcCreate("Your Own", "Doppleganger", <122, 121, 25>, "appearance");
      }
      else if (msg == "create"  && npc != NULL_KEY)     //αν το npc δεν είναι κενό τότε ο
                                                        //npc λέει κάποιο μηνυμα
      {
        osNpcSay(npc, "I am your worst Nightmare!!!!");
      }
                          //αν remove και npc μη κενό διέγραψε μετά από μήνυμα
      else if (msg == "remove" && npc != NULL_KEY)
        osNpcSay(npc, "You will pay for this with your liiiiiivvveeessss!!!.....");
        osNpcRemove (npc);
        npc=NULL_KEY;
      }
      else if (msg == "say" && npc != NULL_KEY)         //αν say ο npc να πεί μήνυμα
```

```
        {
            osNpcSay(npc, "I am your worst Nightmare!!!!");
        }
        else if (msg == "movetarget" && npc != NULL_KEY) //κίνηση πετώντας με προσγείωση
        {
            osNpcMoveToTarget(npc, llGetPos() + <9,9,5>,
OS_NPC_FLY|OS_NPC_LAND_AT_TARGET);
        }
        else if (msg == "movetargetnoland" && npc != NULL_KEY)          //κίνηση πετώντας
        {
            osNpcMoveToTarget(npc, llGetPos() + <9,9,5>, OS_NPC_FLY);
        }
        else if (msg == "movetargetwalk" && npc != NULL_KEY)    //κίνηση περπατώντας
        {
            osNpcMoveToTarget(npc, llGetPos() + <9,9,0>, OS_NPC_NO_FLY);
        }
        else if (msg == "rot" && npc != NULL_KEY)                       //περιστροφή NPC
        {
            vector xyz_angles = <0,0,45>; // This is to define a 45 degree change
            vector angles_in_radians = xyz_angles * DEG_TO_RAD; // Change to Radians
            rotation rot_xyzq = llEuler2Rot(angles_in_radians); // Change to a Rotation
            rotation rot = osNpcGetRot(npc);
            osNpcSetRot(npc, rot * rot_xyzq);
        }
        else if (msg == "save" && npc != NULL_KEY)          //αποθήκευση εμφάνισης NPC
        {
            osNpcSaveAppearance(npc, "appearance");
        }
        else if (msg == "load" && npc != NULL_KEY)//αποθήκευση εμφάνισης AVATAR με key
        {
            osNpcLoadAppearance(npc, "appearance");
        }
        else if (msg == "clone")                //αποθήκευση εμφάνισης κατόχου/owner
        {
            osOwnerSaveAppearance("appearance");
        }
        else if (msg == "stop" && npc != NULL_KEY)          //σταμάτημα κίνησης
        {
            osNpcStopMoveToTarget(npc);
        }
        else if (msg == "destroy")                      //διαγραφή όλων των NPCs στην περιοχή
        {
            list avatars = llList2ListStrided(osGetAvatarList(), 0, -1, 3);
            integer i;
            llSay(0,"NPC Removal: No avatars will be harmed or removed in this process!");
            for (i=0; i<llGetListLength(avatars); i++)
            {
                string target = llList2String(avatars, i);
                osNpcRemove(target);
                llSay(0,"NPC Removal: Target "+target);
```

```
            }
            npc=NULL_KEY;
        }
        else                                    // οποιοδήποτε άλλο μήνυμα εκτελεί το ακόλουθο
        {
            llOwnerSay("I don't understand [" + msg + "]");
        }
    }
  }
}
```

Ανάλυση προγράμματος:

Αρχικά ορίζεται μία σφαιρική/global μεταβλητή με όνομα npc της οποίας η τιμή θα χρησιμοποιηθεί για τον χειρισμό του πράκτορα. Αρχικοποιείται στην τιμή NULL_KEY δηλαδή κενή. Μετά ακολουθεί ο ορισμός της default κατάστασης και το συμβάν state_entry στο οποίο ορίζεται ένα listen συμβάν στο κανάλι δέκα.

Το συμβάν listen χρησιμοποιεί το τμήμα msg για να παραλάβει εντολές από τον χρήστη. Αν πληκτρολογηθεί το μήνυμα create στο κανάλι δέκα και δεν υπάρχει τιμή UUID αναθεμένο στην σφαιρική μεταβλητή npc τότε αρχικά αποθηκεύεται η εμφάνιση του κατόχου του αντικειμένου και δημιουργείται ένας πράκτορας στην θέση <122,121,25> και το UUID κλειδί του αποθηκεύεται στην μεταβλητή npc. Αντίστοιχα αν η npc έχει ήδη κάποια τιμή τότε ο ήδη υπάρχον πράκτορας εμφανίζει το μήνυμα I am your worst nightmare !!!!.Όταν η τιμή του msg είναι και η npc μη κενή τότε ο πράκτορας με UUID ίδιο με την τιμή της npc διαγράφεται αφού πρώτα πει το μήνυμα You will pay for this with your liiiiiivvveeessss!!!.....Αμέσως μετά τίθεται η τιμή της npc σε NULL_KEY ώστε να μπορεί να χρησιμοποιηθεί ξανά η εντολή create.

Όταν το msg έχει την τιμή say και η μεταβλητή npc έχει αναθεμένο κάποιο UUID τότε ο πράκτορας με UUID την τιμή της npc λέει στο βασικό κανάλι I am your worst Nightmare!!!!.Με movetotarget ο πράκτορας πετάει κατά την θέση του συν <9.,9.,5.> και προσγειώνεται μόλις φτάσει στο τέλος της διαδρομής ενώ με movetotargetnoland κάνει το ίδιο αλλά δεν προσγειώνεται. Επίσης με movetargetwalk περπατάει κατά <9.0,9.0,0.0>σε σχέση με την τρέχουσα θέση του. Τέλος αν το msg έχει τιμή rot τότε ο πράκτορας περιστρέφεται κατά 45 μοίρες ως προς τον κατακόρυφο άξονα z'z . Αυτό επιτυγχάνεται ορίζοντας αρχικά ένα διάνυσμα περιστροφής σε μοίρες και μετά μετατροπή του σε radians

ώστε να επιτευχθεί τριγωνομετρικής μορφής του Euler περιστροφή. Αυτή η περιστροφή μετατρέπεται σε quaternion με την llEuler2Rot και μετά εφαρμόζεται μέσω της osNpcSetRot.

Ακολουθεί η δημιουργία καρτέλας εμφάνισης με μία από τις δύο μεθόδους, osNpcSaveAppearance, osOwnerSaveAppearance χρησιμοποιώντας ένα εκ των δύο msg αντίστοιχα save, clone και φόρτωση των στοιχείων από μία καρτέλα με την osNpcLoadAppearance.

Τέλος με destroy δημιουργείται μια λίστα όλων των UUID των πρακτόρων στην περιοχή region και ένας ένας διαγράφονται χωρίς να επηρεαστεί καμία ενσάρκωση. Αντίστοιχα μηνύματα εμφανίζονται και στο βασικό κανάλι επικοινωνίας. Επίσης με stop σταματάει η κίνηση που μπορεί να έχει οριστεί να γίνει ενώ για οποιοδήποτε άλλο msg , το μήνυμα I don't understand συν τα περιεχόμενα του msg περικλυσμένα με αγγύλες.

6.6 Αλλαγή ταχύτητας μετακίνησης

Μετά από την έκδοση 7.2 του Open Simulator παρέχεται η δυνατότητα αύξησης της ταχύτητας μετακίνησης ενός πράκτορα μέσω ενός πολλαπλασιαστή. Η μέθοδος που μας δίνει αυτή την δυνατότητα είναι η osSetSpeed(key npc, float multiplier) η οποία παίρνει σαν ορίσματα το UUID του πράκτορα και έναν πολλαπλασιαστή μέγιστης ταχύτητας μετακίνησης multiplier.

```
key npc=NULL_KEY;
key npc2=NULL_KEY;
key npc3=NULL_KEY;

default
{



  touch_start(integer num)
  {
  npc = osNpcCreate("1.0","", <158, 134, 27>, "appearance");//Δημιουργία npc1
  npc2 = osNpcCreate("1.5","", <158,137,27>, "appearance");//Δημιουργία npc2
  osSetSpeed(npc2,1.5)//ορισμός του συντελεστή της ταχύτητας μετακίνησης στο 1.5
  npc3 = osNpcCreate("0.5","", <158,140,27>,"appearance");//Δημιουργία npc3
  osSetSpeed(npc3,0.5);//ορισμός του συντελεστή της ταχύτητας μετακίνησης στο 0.5
  vector vnpc=osNpcGetPos(npc);          //Εύρεση αρχικής θέσης npc1 και εμφάνιση
  llShout (0,(string)vnpc);
  vector vnpc2=osNpcGetPos(npc2);        //Εύρεση αρχικής θέσης npc2 και εμφάνιση
  llShout (0,(string)vnpc2);
  vector vnpc3=osNpcGetPos(npc3);        //Εύρεση αρχικής θέσης npc3 και εμφάνιση
  llShout (0,(string)vnpc3);             //Έναρξη countdown
  llSleep(3.0);
  llShout(0,"3");
```

```
    llSleep(1.0);
    llShout(0,"2");
    llSleep(1.0);
    llShout(0,"1");
    llSleep(1.0);
    llShout(0,"GO");
                              //Κίνηση
    osNpcMoveToTarget(npc, vnpc + <15,0,0>, OS_NPC_NO_FLY);
    osNpcMoveToTarget(npc2, vnpc2 + <15,0,0>, OS_NPC_NO_FLY);
    osNpcMoveToTarget(npc3, vnpc3 + <15,0,0>, OS_NPC_NO_FLY);

    llSleep(10.0);                          //Αναμονή 10 δευτερολέπτων ώστε οι
                                    //πράκτορες να φτάσουν στον προορισμό τους

    vnpc=osNpcGetPos(npc);          //Ενημέρωση τρέχουσας θέσης του καθενός
    vnpc2=osNpcGetPos(npc2);
    vnpc3=osNpcGetPos(npc3);
                                    //Εμφάνιση των νέων θέσεων
    llShout (0,(string)vnpc);
    llShout (0,(string)vnpc2);
    llShout (0,(string)vnpc3);

    }
}
```

Το παραπάνω script δημιουργεί τρεις πράκτορες με ονόματα τα multipliers στην κίνηση του καθενός και τους κάνει να μετακινηθούν στον άξονα x'x κατά 15 μέτρα σε σχέση με την θέση δημιουργίας τους. Επίσης παρέχει πληροφορίες στο βασικό κανάλι επικοινωνίας για την θέση δημιουργίας τους και για την τελική τους θέση. Λόγω της μη ακριβής μετατόπισης μέσω osNpcMoveToTarget και λόγω ύπαρξης ανωμαλιών στο έδαφος στο τέλος είναι πιθανόν να υπάρχουν αποκλίσεις από τον προορισμό που επιθυμούμε.

Η default ταχύτητα κίνησης περπατώντας είναι περίπου avgspeed=3.799 m/sec. Με κατάλληλη χρήση του multiplier μπορεί να γίνει κίνηση μεγαλύτερης ακριβείας. Όταν κάποιος πράκτορας πρέπει να κινηθεί σε κάποια θέση θα σταματήσει να κινείται σε χρόνο ίσο με t = dist/(avgspeed*multiplier) δευτερόλεπτα αγνοώντας το υπόλοιπο της διαίρεσης.

7.Αλγόριθμοι κίνησης πλήθους

7.1 Εισαγωγή

Το 1986 ο Craig Reynolds πρότεινε ένα μοντέλο συμπεριφορών το οποίο ονόμασε “boids”. Το μοντέλο “boids” προσέφερε έναν τρόπο εξομοίωσης συγχρονισμένων κινήσεων που παρατηρείται σε κοπάδια ψαριών ή πουλιών και που είχε να κάνει με συσχετισμένη κίνηση ως προς τα άλλα μέλη του συνόλου. Το αρχικό μοντέλο ήταν βασισμένο σε τρεις ξεχωριστές και διακριτές συμπεριφορές:

-Διαχωρισμός “Separation”:

Ως σκοπό είχε να προβλέπει και να εμποδίζει τα μέλη ενός κοπαδιού από το να συνοστίζονται σε ορισμένα σημεία του χώρου που καταλαμβάνει το κοπάδι και το αποτέλεσμα του ήταν τα μέλη να διατηρούν μία απόσταση μεταξύ τους.

-Ευθυγράμμιση “Alignment”:

Σκοπός της συμπεριφοράς αυτής ήταν να κάνει τα μέλη να κινούνται προς παράλληλες κατευθύνσεις χρησιμοποιώντας την μέση κατεύθυνση του κοπαδιού και ορίζοντας την κατεύθυνση του κάθε μέλους να τείνει προς την μέση κατεύθυνση.

-Συνοχή “Cohesion”:

Αυτή η συμπεριφορά έβρισκε και κατεύθυνε το κάθε μέλος σε θέση μέσα στο κοπάδι έτσι ώστε να βρίσκεται πάντα στην μέση θέση ως προς τα γειτονικά του μέλη.

Συνδυάζοντας τις παραπάνω συμπεριφορές ο Craig Reynolds κατάφερε να εξομοιώσει ρεαλιστικά την φυσική κίνηση αυτών των φαινομένων. Ένα πρώτο παράδειγμα του “boids” παρουσιάστηκε σε ένα video μικρού μήκους με όνομα “Stanley and Stella in: Breaking the Ice” το οποίο κατασκευάστηκε από το Symbolics Graphics Division σε συνεργασία με την Whitney/Demos Productions και παρουσιάστηκε για πρώτη φορά στο Siggraph 1987.

Αργότερα ο Craig Reynolds στην δημοσίευση του “Not Bumping Into Things” το 1988, παρουσίασε τους υλοποιημένους αλγορίθμους. Επίσης περιέγραψε διάφορους αλγορίθμους και κανόνες για την αποφυγή εμποδίων (Collision Avoidance) με μεθόδους που δεν στηρίζονταν σε πολύπλοκες στρατηγικές και αλγορίθμους εύρεσης μονοπατιού (wayfinding algorithms) αλλά χρησιμοποιώντας δεδομένα από το τοπικό περιβάλλον του κάθε μέλους π.χ. αντικείμενα εντός πεδίου όρασης ή τοπικού χώρου συγκεκριμένης ακτίνας γύρω από το όχημα.

Το 1999 στο συνέδριο Games Developer Conference επέκτεινε και συμπλήρωσε περεταίρω το ανωτέρω μοντέλο. Νέες συμπεριφορές προστέθηκαν οι οποίες ορίζουν συγκεκριμένες αντιδράσεις σε εξομοιώσεις νέων φαινομένων και κινητών. Στο Siggraph 2000, ο Robin Green παρουσίασε μία μελέτη που περιέχει υλοποιήσεις των παραπάνω συμπεριφορών σε γλώσσα C++ καθώς επίσης και βελτιώσεις ορισμένων από αυτών. Επίσης παρουσιάζει προβλήματα τους και προτείνει λύσεις για ορισμένα από αυτά τα προβλήματα. Αυτή η μελέτη έγινε στα πλαίσια του σχεδιασμού και δημιουργίας του παιχνιδιού για υπολογιστές “Dungeon Master 2” από την Bullfrog Productions Ltd.

Παρά το γεγονός ότι οι κατευθυντήριες συμπεριφορές είναι απλές σαν ιδέες η κάθε μία ξεχωριστά , ο συνδυασμός τους μπορεί να αποδώσει ρεαλιστικά πολύπλοκες συμπεριφορές οι οποίες σε αρκετές περιπτώσεις

είναι ικανές να φτάσουν στα άκρα ακόμη και τα πιο καινούρια υπολογιστικά συστήματα της εποχής μας.

Στόχος του κεφαλαίου είναι η αναλυτική επεξήγηση των συμπεριφορών αυτών καθώς και μερικών συνθέσεων τους όπως ορίστηκαν στο παρελθόν από τον Craig Reynolds και τον Robin Green. Επίσης θα παρουσιαστεί η ιδέα υλοποίησης του κάθε αλγορίθμου.

Το μοντέλο κίνησης που πρόκειται να περιγραφεί είναι βασισμένο σε δύο άξονες αν και κατά παρόμοιο τρόπο μπορεί να επεκταθεί και για κίνηση σε τρεις διαστάσεις .

7.2 Βασικές Έννοιες

Για την κατανόηση των συμπεριφορών αλλά και για την υλοποίηση τους πρέπει πρώτα να αναφερθούν ορισμένες έννοιες.

Πιλότος : Pilot

Στον πραγματικό κόσμο κάθε όχημα έχει κάποιον ή κάτι που το κατευθύνει. Ένα αεροπλάνο έχει τον πιλότο , ένα αυτοκίνητο έχει τον οδηγό, ακόμη και ένας άνθρωπος έχει την βούλησή του που τον κατευθύνει. Ομοίως και κατά την αναπαράσταση κίνησης αυτόνομων πρακτόρων υπάρχουν ορισμένα τμήματα κώδικα τα οποία στηριζόμενα σε δεδομένα που μπορεί να προέρχονται από το περιβάλλον των πρακτόρων λαμβάνουν αποφάσεις για το που θα κινηθούν. Κάθε τέτοιο τμήμα κώδικα τεχνητής νοημοσύνης αποτελεί έναν πιλότο.

Όχημα : Vehicle

Ως όχημα ορίζουμε την κάθε αναπαράσταση ενός κινούμενου αντικειμένου. Δεν μας ενδιαφέρει ο τρόπος με τον οποίο εκτελείται η κίνηση ενός οχήματος στον χώρο. Η έννοια του οχήματος εμπεριέχει στοιχεία όπως η θέση του στο χώρο, η ταχύτητά του, η κατεύθυνσή του και άλλες ανάλογα με το πόσο σύνθετο είναι το σύστημά μας.

Για την κατανόηση των συμπεριφορών που ακολουθούν θα χρησιμοποιήσουμε το μοντέλο οχήματος που χρησιμοποίησαν οι Craig Reynolds και Robin Green στις μελέτες τους δηλαδή το σημειακό όχημα. Ένα σημειακό όχημα ορίζεται από την τρέχουσα θέση του και μία προκαθορισμένη μάζα. Επίσης περιλαμβάνει το μέγεθος της ταχύτητας η οποία επηρεάζεται από τις δυνάμεις που επιδρούν στο όχημά μας. Μια εκ των δυνάμεων που μπορεί να ασκηθεί είναι η ώθηση η οποία παράγεται και κατά συνέπεια περιορίζεται σε ρεαλιστικές συνθήκες από την εικονική μηχανή σε συγκεκριμένη και προκαθορισμένη μέγιστη τιμή. Ομοίως υπάρχει μια προκαθορισμένη τιμή στην ταχύτητα την οποία το όχημα δεν μπορεί ποτέ να ξεπεράσει. Αυτή η μέγιστη ταχύτητα αντιπροσωπεύει την τιμή που επιτρέπουν εξωτερικοί παράγοντες στον πραγματικό κόσμο όπως η τριβή και η αντίσταση του περιβάλλοντος στην κίνηση. Τέλος ένα σημειακό όχημα χαρακτηρίζεται από την κατεύθυνσή του η οποία γενικά υποδηλώνει προς τα πού κινείται το όχημα και αναπαρίσταται με διάνυσμα.

Απλό Σημειακό Όχημα

- Μάζα - Σταθερά

- Θέση - Διάνυσμα από το σημείο αναφοράς του χώρου

- Ταχύτητα - Διάνυσμα έως το σημείο που θα βρίσκεται μία χρονική στιγμή μετά το όχημα από την τρέχουσα θέση του

- Μέγιστη Δύναμη - Τιμή που δεν μπορεί να ξεπεράσει ποτέ η ώθηση του οχήματος σαν μέτρο

- Μέγιστη Ταχύτητα- Τιμή που δεν μπορεί να ξεπεράσει ποτέ η ταχύτητα του οχήματος σαν μέτρο

- Κατεύθυνση - Διάνυσμα που υποδηλώνει προς τα που κινείται το όχημα

Για συγκεκριμένη μονάδα χρόνου λοιπόν μπορούμε να υπολογίσουμε την μελλοντική θέση ενός οχήματος ακολούθως:

position=position+current_velocity;

Δηλαδή η νέα θέση θα είναι ίση με την τρέχουσα θέση συν την ταχύτητα.

Ομοίως η ταχύτητα θα είναι:

velocity=velocity+acceleration

Δηλαδή η νέα ταχύτητα θα είναι ίση με την τρέχουσα ταχύτητα συν την επιτάχυνση που θα προκύψει από το άθροισμα των δυνάμεων. Πρέπει όμως να αναφερθεί ότι το μέτρο της νέας ταχύτητας που θα προκύψει δεν πρέπει να ξεπερνάει την τιμή της μέγιστης ταχύτητας (max_speed). Επίσης η μέγιστη επιτάχυνση δεν μπορεί να ξεπερνάει σε μέτρο την τιμή max_acceleration.

Για ελαχιστοποίηση σε αριθμό και πολυπλοκότητα των πράξεων θα χρησιμοποιήσουμε ένα τοπικό σύστημα συντεταγμένων στο οποίο η ταχύτητα του οχήματος αντιστοιχεί στον άξονα x΄x ώστε να μην χρειάζεται να κάνουμε μετασχηματισμό όλων των διανυσμάτων που επιδρούν στο όχημά μας ξεχωριστά. Για να το επιτύχουμε αυτό περιστρέφουμε τον άξονα x΄x ώστε να συμπίπτει με το διάνυσμα της ταχύτητας. Έτσι θα έχουμε:

new_forward = normalize(current_velocity)

approximate_up = normalize(approximate_up)

new_side = cross(new_forward,approximate_up)

new_up=cross(new_forward,new_side)

Η βασική ιδέα είναι ότι το approximate_up είναι προσεγγιστικά κάθετο στο new_forward καθώς στην κίνηση ανά πολύ μικρά χρονικά διαστήματα οι μεταβολές είναι πολύ μικρές.

Συμπεριφορά : Behavior

Ο όρος συμπεριφορά είναι πιθανόν να πάρει πολλές έννοιες. Μπορεί να πρόκειται για μια σύνθετη ενέργεια κάποιου ανθρώπου ή κάποιου άλλου ζώου βασισμένη στην βούληση ή ένστικτο του. Η σύνθετη ενέργεια ενός χαοτικού συστήματος μπορεί επίσης να χαρακτηριστεί ως συμπεριφορά ή ακόμη και η προσδοκώμενη λειτουργία μιας μηχανής. Πολύ συχνά ο όρος αυτός χρησιμοποιείται για το animation ενός χαρακτήρα σε έναν εικονικό κόσμο. Σε αυτήν εδώ την μελέτη θα αντιπροσωπεύει τις ρεαλιστικές ενέργειες που εκτελεί ένας αυτόνομος χαρακτήρας.

Όπως και στην πραγματική ζωή δεν κινούμαστε πάντα κατά τον ίδιο τρόπο για να φτάσουμε στο επιθυμητό αποτέλεσμα. Το ίδιο προσπαθούμε να επιτύχουμε και στους αυτόνομους χαρακτήρες μας. Αν παραδείγματος χάρη κάποιος περπατήσει μεταξύ εμάς και του στόχου μας , είναι πολύ πιθανόν να επιλέξουμε να αποφύγουμε το εμπόδιο ακολουθώντας διαφορετική πορεία ώστε να αποφύγουμε μια πιθανή σύγκρουση. Η συμπεριφορά ενός αυτόνομου πράκτορα κατά τον Craig Reynolds μπορεί να χωριστεί σε τρία επίπεδα:

- Επιλογή ενέργειας : Στρατηγική, Στόχοι, Σχεδιασμός

- Κατεύθυνση : Καθορισμός Διαδρομής

- Κίνηση : animation

Παράδειγμα που μπορεί να μας δώσει μια εικόνα για τον ανωτέρω διαχωρισμό είναι η διαδρομή που κάνει κάποιος όταν μπαίνει σε ένα ταξί. Ο πελάτης λέει στον οδηγό την διεύθυνση που θέλει να πάει. Ο οδηγός τότε ακολουθεί μια σειρά ενεργειών για να φτάσει στον προορισμό αποφεύγοντας την κίνηση και τα εμπόδια καθώς επίσης και σταματάει στα φανάρια. Ο πελάτης μπορεί να χαρακτηριστεί ως επιλογέας ενέργειας και δίνει στον οδηγό έναν στόχο (διεύθυνση) ή ακόμη και σχεδιασμό αν παραδείγματος χάρη ζητήσει απο τον οδηγό να σταματήσει κάπου και να περιμένει ή να ακολουθήσει μια συγκεκριμένη εκ των πιθανών διαδρομών. Ο οδηγός αποτελεί την κατεύθυνση ως επίπεδο του παραδείγματος. Είναι αυτός ο οποίος αναλύει τον στόχο σε περισσότερους πιο βραχυπρόθεσμους στόχους και που εκτελεί τις κατάλληλες ενέργειες ώστε να φτάσει στον επιθυμητό από τον πελάτη προορισμό. Τέτοιου τύπου ενέργειες μπορούν να εκτελούνται από τον οδηγό δίνοντας εντολή στο αυτοκίνητο να τις εκτελέσει στρίβοντας το τιμόνι , πατώντας το γκάζι η ελαττώνοντας την ταχύτητα με την χρήση του φρένου και αποτελούν απλές λειτουργίες όπως πήγαινε δεξιά η αριστερά αύξησε την ταχύτητα η πήγαινε πιο αργά. Ο τρόπος κίνησης ανήκει στο αυτοκίνητο το οποίο χρησιμοποιώντας την μηχανή η τα φρένα του εκτελεί τις επιμέρους ενέργειες.

Παρόλο που η κίνηση καθεαυτού αποτελεί σοβαρό θέμα στον τομέα του animation, δεν πρόκειται να εξεταστεί σε αυτήν εδώ την μελέτη καθώς ο σκοπός της είναι η ανάλυση του δευτέρου επιπέδου συμπεριφοράς κατά κύριο λόγο.

Επίσης όπως αναφέρθηκε νωρίτερα υπάρχουν τμήματα κώδικα τα οποία έχουν ως σκοπό την εκπλήρωση τέτοιων λειτουργιών όπως η λήψη αποφάσεων για την κίνηση ενός οχήματος. Τέτοια τμήματα κώδικα τα ορίζουμε ως συμπεριφορές μέσα στο πρόγραμμά μας λαμβάνουν αποφάσεις για το εκάστοτε όχημα βασισμένα στις τρέχουσες συνθήκες

με σκοπό την παραγωγή μιας δύναμης ικανής να επηρεάσει καταλλήλως το όχημά μας προς το επιθυμητό αποτέλεσμα.

7.3 Κατευθυντήριες συμπεριφορές

Ο Craig Reynolds περιέγραψε και ανέλυσε ορισμένες συμπεριφορές που προκαλούν φυσιολογική κίνηση στους πράκτορες. Χρησιμοποίησε για μοντέλα , φαινόμενα τα οποία παρατηρούνται σε καθημερινές καταστάσεις τα οποία ύστερα από ανάλυση , παρατήρησε ότι μπορούν να αναπαρασταθούν μέσω κάποιας συμπεριφοράς ή κάποιου συνδυασμού αυτών. Για την περιγραφή τους χρησιμοποιήθηκε το μοντέλο του σημειακού οχήματος σε δισδιάστατο χώρο αλλά δεν είναι δύσκολο να γίνουν μετατροπές ώστε να επεκταθούν οι παραπάνω συμπεριφορές και στον τρισδιάστατο χώρο. Επίσης το μέτρο των ασκούμενων δυνάμεων συνήθως δεν αποτελεί πρόβλημα από πλευράς υπολογισμών στο σύστημα καθώς η μέγιστη δύναμη θα περιορίζεται στην τιμή της μέγιστης τιμής που μπορεί να ασκηθεί στο όχημα(max force).

Οι βασικές συμπεριφορές καθώς επίσης και η υλοποίηση μερικών από αυτές σε LSL ακολουθούν.

7.3.1 Seek και Flee

Η συμπεριφορά seek είναι αυτή κατά την οποία το όχημα προσπαθεί να περάσει από κάποιο σημείο προορισμού. Κάθε κάποιο ορισμένο χρονικό διάστημα γίνεται έλεγχος της ταχύτητας και της θέσης του οχήματος καθώς και της θέσης του στόχου και εκτελείται ο αλγόριθμος

που ακολουθεί . Αφού γίνει η αφαίρεση των διανυσμάτων της θέσης του οχήματος από την θέση του στόχου , γίνεται κανονικοποίηση του διανύσματος που προκύπτει και μετά αυτό πολλαπλασιάζεται με την μέγιστη επιτρεπτή ταχύτητα. Αυτή η διαδικασία επιστρέφει την επιθυμητή ‘’ρεαλιστική’’ ταχύτητα που το κινητό μπορεί να πιάσει. Η επιτάχυνση που πρέπει να υπάρχει για να επιτευχθεί αυτό είναι ίση με την διαφορά της επιθυμητής ταχύτητας μείων την τρέχουσα ταχύτητα του κινητού. Στο σημειακό όχημα η μάζα είναι μια μονάδα οπότε ο τύπος F=m*γ ορίζει ότι η δύναμη ισούται με την επιτάχυνση. Έτσι έχοντας ορίσει την μέγιστη δύναμη, περικόπτεται η κατευθυντήρια δύναμη ώστε να γίνει ίση σε μέτρο με την μέγιστη επιτρεπτή. Σε κίνηση στην οποία η μάζα του κινητού είναι διαφορετική υπολογίζεται πρώτα η δύναμη που πρέπει να ασκηθεί από τον παραπάνω τύπο και ύστερα περικόπτεται.

Έτσι για το σημειακό όχημα

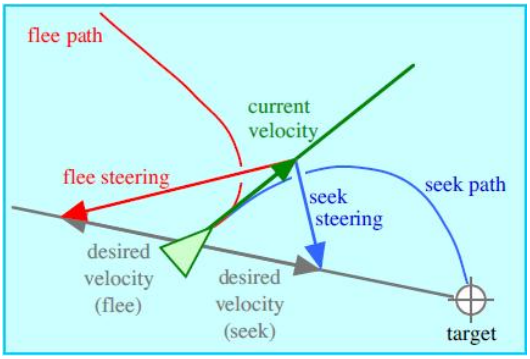
desired_velocity = norm(target-position)*max_speed;

desired_force= desired_velocity-current_velocity;

applied_force=prune(desired_force);

new_velocity=current_velocity+applied_force;

Η συμπεριφορά Flee έχει το ακριβώς αντίθετο αποτέλεσμα από την seek. Αυτή η συμπεριφορά οδηγεί το όχημα όσο το δυνατόν συντομότερα μακριά από το σημείο στόχου. Η διαφορά της κατά την υλοποίηση είναι ότι η διεύθυνση του διανύσματος παραγωγής επιθυμητής ταχύτητας είναι αντίθετη από την seek. Έτσι



desired_velocity=norm(position-target)*max_speed;

7.3.2 Pursuit και Evasion.

Πρόκειται για μία ακόμη παραλλαγή της Seek. Πλέον ο

στόχος κινείται επίσης. Η λογική είναι ίδια απλά σε κάθε έλεγχο ορίζουμε στην seek ένα νέο σημείο στόχου το οποίο βρίσκεται στη θέση που αντιστοιχεί στην θέση του οχήματος στόχου μετατοπισμένο κατά την τρέχουσα ταχύτητά του επί το χρονικό διάστημα που παρεμβάλλεται μεταξύ των ελέγχων. Έτσι το αρχικό όχημα ‘’προβλέπει’’ την συμπεριφορά του οχήματος στόχου και κινείται προς αυτήν την θέση. Ο χρόνος μεταξύ των ελέγχων όταν είναι σταθερός και μικρός, δίνει τα πιο ρεαλιστικά αποτελέσματα ωστόσο όταν καλείται το σύστημα να επεξεργαστεί πολλές παρόμοιες συμπεριφορές είναι επιθυμητό να περιορίζονται οι πράξεις που εκτελούνται από το σύστημα. Μια άλλη προσέγγιση του θέματος θα μπορούσε να είναι ο έλεγχος να γίνεται ανάλογα με την απόσταση την οποία βρίσκεται ο στόχος. Όταν είναι μακριά ο έλεγχος γίνεται πιο αραιά καθώς επίσης και η πρόβλεψη για την μελλοντική θέση του στόχου αλλάζει ενώ όταν η απόσταση μικραίνει τότε ο έλεγχος γίνεται πιο συχνά ώστε να υπάρχει μεγαλύτερη ακρίβεια.

Παρομοίως το evasion , όπως και το flee έχει ακριβώς την αντίθετη συμπεριφορά. Το όχημα στο οποίο εφαρμόζεται προσπαθεί να αποφύγει το όχημα στόχο βρίσκοντας την πιο άμεση διαδρομή. Η διαφορά των δύο συμπεριφορών βρίσκεται στην διεύθυνση της επιθυμητής ταχύτητας.

7.3.3 Arrival

Ο αλγόριθμος seek έχει ένα βασικό μειονέκτημα. Ο τρόπος με τον οποίο είναι φτιαγμένος φροντίζει ώστε το όχημα στο οποίο εφαρμόζεται να περάσει από κάποιο σημείο αλλά δεν υπάρχει έλεγχος της ταχύτητας με την οποία περνάει από το σημείο αυτό. Συνέπεια αυτής της συμπεριφοράς είναι ότι το όχημα περνάει από το σημείο αυτό και αμέσως μετά έχοντας την ταχύτητα αυτή απομακρύνεται και πάλι. Όμως ο αλγόριθμος συνεχίζει να εκτελείται και έτσι αμέσως μετά αρχίζει να επιβραδύνει και προσπαθεί να επιστρέψει στο συγκεκριμένο σημείο. Το φαινόμενο που παρουσιάζεται είναι παρόμοιο με μια πεταλούδα που τριγυρνάει γύρω από μια λάμπα. Παρομοίως και στην pursuit παραλλαγή το όχημα προσπερνάει το όχημα που κυνηγάει και επιστρέφει ξανά.

Η βασική διαφορά του arrival είναι ότι η ταχύτητα του οχήματος στο οποίο εφαρμόζεται προσαρμόζεται ανάλογα με το πόσο κοντά βρίσκεται στο στόχο. Ορίζεται μία απόσταση κατά τον προγραμματισμό τέτοια ώστε όσο το όχημα κινείται εκτός αυτής η μέγιστη ταχύτητα του είναι αυτή που έχει προκαθοριστεί κατά τον ορισμό του οχήματος. Όταν το όχημα μετακινείται σε σημείο το οποίο απέχει μικρότερη απόσταση από αυτή την απόσταση ασφαλείας αρχίζει να επιβραδύνει ώστε να φτάσει στον προορισμό του με μηδενική ταχύτητα.

Αντίστοιχα σε παραλλαγή pursuit το όχημα μόλις φτάσει σε σχεδόν μηδενική απόσταση από το όχημα στόχος τότε η μέγιστη ταχύτητα του τείνει στο μέτρο της τρέχουσας ταχύτητας του οχήματος στόχου. Αν και οπτικά για τον χρήστη το αποτέλεσμα είναι ρεαλιστικό στην πραγματικότητα το όχημα που εφαρμόζει arrival ή συνδυασμό pursuit και arrival ποτέ δεν φτάνει στο στόχο. Επίσης ορισμένες φορές ο προσανατολισμός του σημειακού οχήματος αλλάζει σε μία φυσικές κατευθύνσεις.

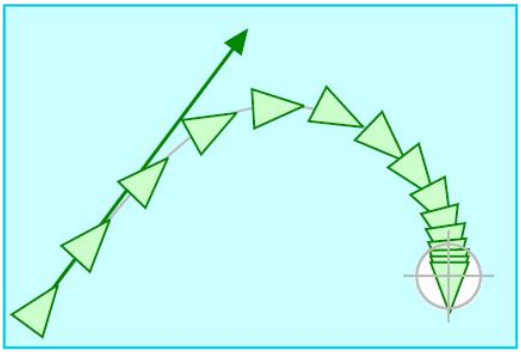
Η λογική του arrival είναι η ακόλουθή. Αρχικά ορίζεται η απόσταση ασφαλείας. Όσο το όχημα στο οποίο εφαρμόζεται είναι εκτός αυτής της απόστασης εφαρμόζεται ο seek αλγόριθμος. Όταν το όχημα μπει εντός της απόστασης ασφαλείας τότε περικόπτεται η μέγιστη ταχύτητα.

offset=target-position;

distance=magnitude(offset);

ramped_speed = max_speed*(distance/safe_distance);

Η παραπάνω έκφραση μεταβάλει την τιμή της ramped_speed από μικρότερη από max_speed έως πολύ μεγαλύτερη. Για να εξασφαλιστεί ότι το όχημα θα κινηθεί το πολύ με την max_speed άλλα δεν θα έχει μεγαλύτερη από την ταχύτητα που πρέπει βάσει την απόσταση που απέχει από τον στόχο βρίσκεται η μικρότερη τιμή από τις δύο και το όχημα ορίζεται να κινηθεί με αυτήν.



clipped_speed = min(max_speed,ramped_speed);

Η επιθυμητή ταχύτητα θα είναι ίση με την clipped_speed επί την κατεύθυνση της απόστασης μεταξύ του οχήματος και του στόχου. Για να βρεθεί αυτή η ταχύτητα αρκεί να κανονικοποιηθεί το διάνυσμα. Εδώ μπορούν να αποφευχθούν πράξεις ώστε να χρησιμοποιείται λιγότερη υπολογιστική ισχύ σε κάθε επανάληψη του ελέγχου αν επαναχρησιμοποιηθούν τα δεδομένα ώστε να γίνει κανονικοποίηση του διανύσματος offset χωρίς να χρειαστεί να ξαναυπολογιστεί το μέτρο του.

Έτσι προκύπτει…

desired_velocity = clipped_speed*(offset/distance);

και

steering=desired_velocity-current_velocity;

7.3.4 Offset Pursuit

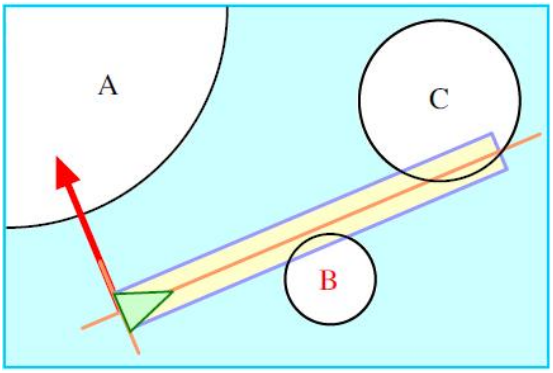
Στην περίπτωση πού χρησιμοποιούνται σημειακά οχήματα τότε είναι εσκεμμένο ο στόχος να είναι η θέση του σημείου τελικού προορισμού. Όμως αν χρησιμοποιηθεί ως προσομοίωση μια κατάσταση όπως ένα αυτοκίνητο που προσπαθεί να φτάσει κοντά σε κάποιο κτίριο τότε καθώς το αυτοκίνητο και το κτίριο καταλαμβάνουν κάποιο χώρο, το αυτοκίνητο θα πρέπει να παρκάρει σε κάποια απόσταση από το κέντρο του κτιρίου στις περισσότερες περιπτώσεις. Ομοίως ένα αυτοκίνητο που προσπαθεί να φτάσει κάποιο άλλο θα πρέπει όταν φτάσει στον στόχο να συνεχίσει να κινείται σε κάποια απόσταση από την σχετική θέση του οχήματος στόχου. Αυτό μπορεί να επιτευχθεί με μια παραλλαγή του arrival και του pursuit. Αυτή η παραλλαγή ονομάζεται Offset Pursuit.

7.3.5 Obstacle Avoidance

Μια ακόμη συμπεριφορά που παρουσιάζει ιδιαίτερο ενδιαφέρον είναι η αποφυγή εμποδίων. Σε περίπτωση που το όχημα πρόκειται να συγκρουστεί με κάποιο εμπόδιο αν συνεχίσει την κίνηση του η συμπεριφορά αυτή ορίζει προς τα πού θα πρέπει να κινηθεί για να

αποφευχθεί κάτι τέτοιο. Ωστόσο σε αντίθεση με τον αλγόριθμο flee που το όχημα αποφεύγει να βρίσκεται κοντά σε αντικείμενο ή θέση στόχο και που στο τέλος θα καταλήξει με ταχύτητα κάθετη ως προς αυτό, ο αλγόριθμος Obstacle Avoidance ενεργεί μόνο όταν κάτι βρίσκεται ακριβώς μπροστά στην κατεύθυνση κίνησης του οχήματος ενώ θα αγνοήσει οποιοδήποτε άλλο εμπόδιο δεν παρεμβάλλεται στην κίνησή του.

Ο αλγόριθμος που έχει προταθεί από τον Craig Reynolds για λόγους απλοποίησης θεωρεί ως δεδομένο ότι τόσο τα εμπόδια όσο και το όχημα μπορούν να αναπαρασταθούν προσεγγιστικά ως σφαίρες ή κύκλοι σε επίπεδο δύο διαστάσεων. Αυτό αποτελεί συχνά πρόβλημα βέβαια καθώς τόσο τα οχήματα όσο και τα εμπόδια συχνά δεν μπορούν να αναπαρασταθούν ρεαλιστικά ως σφαίρες. Παραδείγματος χάρη ενώ αυτό το σύστημα είναι καλό όταν σκοπός είναι ένας πεζός να αποφύγει ένα δέντρο , είναι κακή προσέγγιση όταν ένας στενός σχετικά δρόμος χωρίζεται σε 2 ίσα μέρη από έναν τοίχο παράλληλο προς τον δρόμο. Στην δεύτερη περίπτωση το όχημα θα βγει εκτός δρόμου.



Για την αναπαράσταση αυτού πρέπει να οριστεί αρχικά η ακτίνα της σφαίρας που ορίζει το όχημα. Αποφυγή σύγκρουσης συνεπάγεται ότι η διαδρομή μπροστά από το όχημα είναι καθαρή από εμπόδια. Αρκεί να οριστεί ένας κύλινδρος μπροστά από το όχημα που θα καθορίζει ποία περιοχή πρέπει να ελέγχεται για τυχόν εμπόδια. Το μήκος του κυλίνδρου ορίζει πότε ένα αντικείμενο ενώ βρίσκεται μπροστά από την τρέχουσα κατεύθυνση του οχήματος δεν αποτελεί άμεσο κίνδυνο για το όχημα και εξαρτάται από την ταχύτητα και την δυνατότητα αλλαγής κατεύθυνσης του οχήματος.

Αρχικά μεταφέρουμε το σύστημα σε τοπικές συντεταγμένες και προβάλουμε τα κέντρα των εμποδίων στον άξονα x'x. Εφόσον το κέντρο των αξόνων του συστήματος είναι το κέντρο του οχήματος , όποια προβολή αντικειμένου βρίσκεται στο αρνητικό τμήμα του τοπικού άξονα

x’x, ορίζει ότι το αντικείμενο αυτό αγνοείται καθώς βρίσκεται πίσω από το όχημα. Παρομοίως κάθε αντικείμενο του οποίου η προβολή από το κέντρο του βρίσκεται σε απόσταση μεγαλύτερη του μήκους του κυλίνδρου αγνοείται διότι δεν αποτελεί άμεση απειλή.

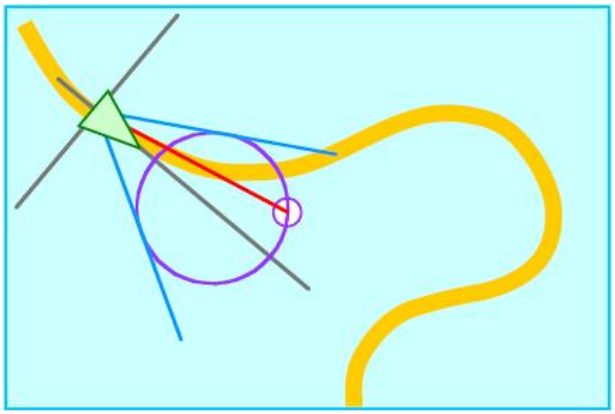
Αν η προβολή βρίσκεται εντός της εμβέλειας ελέγχου τότε ελέγχεται το μήκος της προβολής και αν είναι μικρότερο ή ίσο με το άθροισμα της ακτίνας του οχήματος και της ακτίνας του αντικειμένου τότε το όχημα πρόκειται να συγκρουστεί με αυτό το εμπόδιο οπότε μπορεί να ασκηθεί σε αυτό μια δύναμη κάθετη που θα το κάνει να αλλάξει πορεία προς την αντίθετη κατεύθυνση από το κέντρο του συγκεκριμένου εμποδίου. Η δύναμη που ασκείται θα μπορούσε να είναι ανάλογη με την διαφορά του μήκους της προβολής ως προς το άθροισμα των ακτινών. Πάντα επιλέγεται το κοντινότερο εμπόδιο από όλα τα πιθανά ως το πλέον άμεσα επικίνδυνο και με βάση αυτό αποφεύγονται τα εμπόδια.

7.3.6 Wander

Η τυχαία κίνηση ενός οχήματος αρχικά φαίνεται δύσκολη στην εξομοίωση. Σαν σύλληψη δεν έχει να κάνει με κάποιον συγκεκριμένο στόχο. Η πιο εύκολη λύση για τυχαία κίνηση πιθανώς κάποιος να φανταζόταν ότι θα είναι η παραγωγή ενός τυχαίου διανύσματος. Δυστυχώς αυτή η μέθοδος δεν προκαλεί «φυσική» κίνηση. Για την κατανόηση του προβλήματος θα χρησιμοποιηθεί το παράδειγμα ενός περιπάτου σε ένα πάρκο. Δεν είναι υποχρεωτικό λοιπόν να ακολουθηθούν τα μονοπάτια και μπορεί κανείς να πάει όπου θέλει μέσα σε αυτό. Αυτό είναι ένα παράδειγμα τυχαίας κίνησης όταν δεν υπάρχει συγκεκριμένος προορισμός.

Ο άνθρωπος που περπατάει, κινείται , στρίβει, επιταχύνει , επιβραδύνει ανάλογα με το τι του αρέσει εκείνη την στιγμή. Όμως ότι και αν γίνει δεν μπορεί την μία χρονική στιγμή να πηγαίνει προς μια κατεύθυνση και την επόμενη να κινείται προς αντίθετη κατεύθυνση. Ούτε μπορεί να αλλάξει κατεύθυνση προς γωνία μεγαλύτερη από ότι του επιτρέπει το σώμα του. Αν λοιπόν εφαρμοζόταν η μέθοδος που

αναφέρθηκε νωρίτερα και το τυχαίο διάνυσμα της μιας στιγμής ήταν αντίθετο σε φορά από ότι το διάνυσμα της επόμενης η κίνηση που θα προέκυπτε δεν θα ήταν φυσική. Είναι λοιπόν λογικό το συμπέρασμα ότι η τυχαία κίνηση κάποιου στο πάρκο δεν είναι και τόσο τυχαία αλλά εξαρτάται από την τρέχουσα κατεύθυνση του και την δυνατότητα στροφής.



Για να γίνει κάτι τέτοιο σε έναν εικονικό κόσμο αρκεί να εξασφαλιστεί αυτή η εξάρτηση. Αυτό μπορεί να επιτευχθεί αν σε κάθε χρονική στιγμή που ορίζουμε ότι μπορεί να αλλάξει η κίνηση σκεφτεί κανείς την νέα κίνηση σαν ένα άθροισμα ενός σταθερού μέτρου διανύσματος πού ταυτίζεται με την τρέχουσα ταχύτητα του οχήματος σε διεύθυνση (όχι και σε μέτρο απαραίτητα) και ενός διανύσματος πού ξεκινάει από το πέρας του σταθερού διανύσματος και του οποίου το πέρας βρίσκεται στην περιφέρεια ενός κύκλου τέτοιο ώστε η ακτίνα του κύκλου και το μέτρο του σταθερού διανύσματος να μην ξεπερνούν κατά προτίμηση την max_speed. Με αυτόν τον τρόπο επιτυγχάνεται τόσο τυχαία φορά κίνησης , αν και πάντα σχετική με την ταχύτητα την αμέσως προηγούμενη στιγμή, όσο και ο περιορισμός της δυνατότητας στροφής του οχήματος παρόμοιας με την δυνατότητα στροφής κάποιου πραγματικού οχήματος.

7.3.6.1 Υλοποίηση σε LSL και επεξήγηση

Κατά την υλοποίηση του αλγορίθμου και προσπαθώντας να κατασκευαστεί κάτι το οποίο θα μπορούσε να αποδώσει καλύτερα από ότι ο αρχικός αλγόριθμος με σκοπό είσοδο περισσοτέρων πρακτόρων ταυτόχρονα προέκυψε μια παραλλαγή του. Αυτή η παραλλαγή θεωρώντας ότι ο πράκτορας θα κινείται σε μόνιμη βάση με την μέγιστη ταχύτητα που του έχει οριστεί προκαλεί τυχαία κίνηση επιλέγοντας ένα διάνυσμα που ανήκει στο εμβαδό που ορίζει ένα τετράγωνο του οποίου το κέντρο απέχει κατά κάποια συγκεκριμένη απόσταση από τις πλευρές του αντί για το διάνυσμα που ανήκει στην περιφέρεια του κύκλου. Η

απόσταση αυτή είναι το μισό της εμβέλειας της συνάρτησης llFrand για randx.

Επίσης για αποφυγή μετατροπών από global σε local συντεταγμένες συνεχώς χρησιμοποιείται η παρατήρηση ότι αν η νέα κατεύθυνση κάθε φορά είναι η φορά της κίνησης του πράκτορα τότε γνωρίζοντας αυτή την φορά μπορεί εύκολα να υπολογιστεί ποιο θα είναι το διάνυσμα που θα ανήκει στον τοπικό άξονα y'y. Είναι γνωστό ότι

$\sin(\pi/2 + x) = -\cos(x)$

και

$\cos(\pi/2 + x) = \sin(x)$

Άρα κανονικοποιώντας το διάνυσμα της διεύθυνσης κίνησης offset μπορεί εύκολα να βρεθεί το κάθετό του το οποίο θα είναι το <-xvec.y, xvec.x, 0> αφού η κίνηση γίνεται σε δύο διαστάσεις. Έτσι προκύπτει…

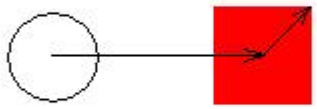
Αρχικά ορίζονται οι μεταβλητές…

```
vector xvec = <1.0,0.0,0.0>;
vector yvec = <0.0,1.0,0.0>;
vector offset;
float randx;
float randy;
float length;
```

Αμέσως μετά κατασκευάζεται η μέθοδος υπολογισμού της θέσης στην οποία ο πράκτορας θα πρέπει να μετακινηθεί η οποία κατά το τέλος της κλήσης της ορίζει το νέο διάνυσμα που αντιστοιχεί στον άξονα x'x και το διάνυσμα που θα αντιστοιχεί στον νέο άξονα y'y. Εδώ χρησιμοποιείται η συνάρτηση llFrand(float range) με σκοπό να οριστεί η τυχαιότητα της διεύθυνσης και του μήκους τις απόκλισης από την αρχική θέση. Με διαίρεση του μήκους του διανύσματος απόκλισης offset προς την μέση ταχύτητα κίνησης στο έδαφος ορίζεται ένας συντελεστής αυτής στην μέγιστη ταχύτητα κίνησης.

```
wander()
{
   randx=(llFrand(1.6)+2.2);
   randy=(llFrand(1.6)-0.8);
   offset=randx*xvec+randy*yvec;
```

```
   length=llVecMag(offset);
   osSetSpeed(length/3.8);

osNpcMoveToTarget(llGetOwner(),llGetPos()+offs
et,OS_NPC_NO_FLY);

xvec=<offset.x/length,offset.y/length,offset.z/leng
th>;
   yvec=<-xvec.y,xvec.x,0.0>;

}
```



Η κατάσταση default ορίζεται να έχει ένα συμβάν listen στο κανάλι 92 με είσοδο για αυτό το συμβάν την λέξη wander. Όταν αυτό το συμβάν εκτελείται το σύστημα μεταβαίνει στην κατάσταση wander.

```
default
{
  state_entry()
  {
     llListen(92,"","","wander");

  }
  listen(integer chan,string name,key id,string msg)
  {
          state wander;
  }
}
```

Η κατάσταση wander κατά την είσοδο σε αυτήν εκτελεί την συνάρτηση wander() και εκκινεί ένα συμβάν timer όπου εκτελείται κάθε 0.5 δευτερόλεπτα και αναμένει στο κανάλι 92 την εντολή stop η οποία σταματάει το συμβάν timer και επιστρέφει στην κατάσταση default. Στο συμβάν timer εκτελείται η μέθοδος wander κάθε φορά που αυτό επαναλαμβάνεται.

```
state wander
{
  state_entry()
  {
     wander();
```

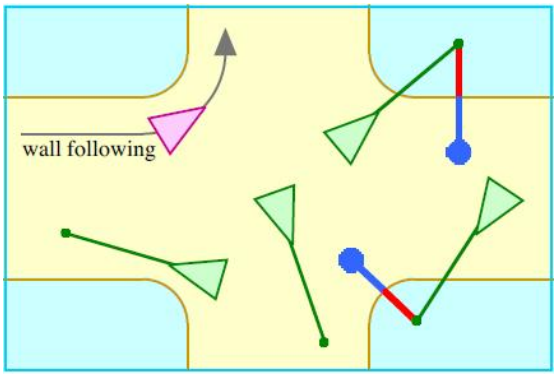
```
        llSetTimerEvent(0.5);
        llListen(92,"","","stop");

    }
    timer()
    {
        wander();

    }
    listen(integer chan,string name,key id,string msg)
    {
        llSetTimerEvent(0.0);
        state default;
    }
}
```

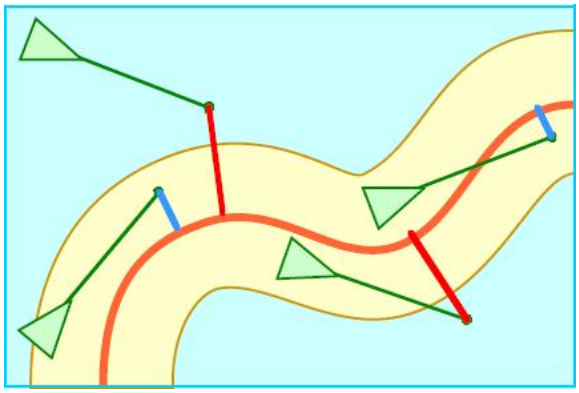
Ο OpenSimulator έχει προκαθορισμένη ταχύτητα μετακινήσεων κατά συνέπεια κινείται όσο προλαβαίνει σε κάθε χρονική περίοδο που εκτελείται η μέθοδος wander() οπότε αν το offset είναι μεγαλύτερο από ότι προλαβαίνει να κινηθεί ο πράκτορας και ενημερώνεται κάθε φορά που εκτελείται η μέθοδος , επιτρέπει την συνεχή κίνηση με μεταβλητή ταχύτητα προς την σωστή διεύθυνση κάθε φορά.

Τέλος , απαιτείται ένα αντικείμενο το οποίο θα δημιουργεί τους πράκτορες σε κάποια θέση και ένα αντικείμενο ακόμη το οποίο είναι προσαρτημένο σε κάποιο μέλος του σώματος του κάθε πράκτορα το οποίο θα περιέχει τον παραπάνω κώδικα ώστε να εκτελείται διαρκώς ο κώδικας στον εκάστοτε πράκτορα.

7.3.7 Path Following, Wall Following ,Containment και Flow Field Following

Ο αλγόριθμος αυτός επιτρέπει κίνηση εντός αποδεκτών ορίων κάποιου προκαθορισμένου μονοπατιού. Πρόκειται για ένα μονοπάτι το οποίο είναι προκαθορισμένο και μία ακτίνα r η οποία είναι η αποδεκτή απόκλιση από το μονοπάτι. Αρχικά ο αλγόριθμος προβλέπει την μελλοντική θέση του οχήματος με βάση την τρέχουσα ταχύτητα. Αν η απόσταση της μελλοντικής θέσης από το μονοπάτι είναι μικρότερη ή ίση με την ακτίνα τότε το όχημα συνεχίζει να κινείται προς την κατεύθυνση της ταχύτητάς του. Αν η μελλοντική θέση του οχήματος είναι σε απόσταση μεγαλύτερη από την ακτίνα r τότε ο αλγόριθμος seek εκτελείται για το σημείο πάνω στο μονοπάτι το οποίο αντιστοιχεί στην προβολή της μελλοντικής θέσης που υπολογίστηκε.



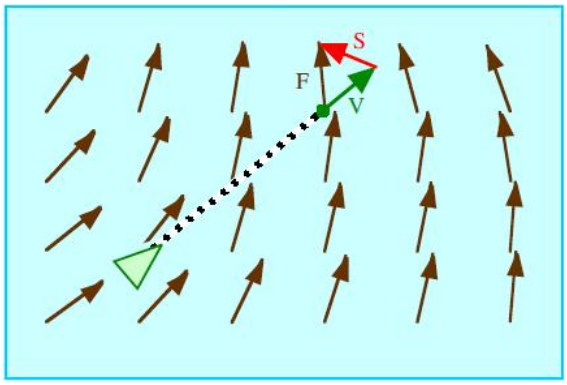


Μια παραλλαγή του Path Following είναι το Wall Following το οποίο χρησιμοποιεί την γραμμή κάποιου τοίχου και προσπαθεί να διατηρήσει κάποια απόσταση(offset) από αυτήν με τρόπο ανάλογο με το offset pursuit.

Το Path Following είναι μια μορφή του αλγορίθμου containment όταν ο χώρος κίνησης του οχήματος είναι ένας κύλινδρος μπροστά από το όχημα. Παράδειγμα περίπτωσης containment θα μπορούσε να είναι η κίνηση ψαριών σε ένα ενυδρείο. Για να επιτευχθεί η αναπαράσταση ενός τέτοιου φαινομένου αρχικά υπολογίζεται η μελλοντική θέση του οχήματος. Αν αυτή είναι εντός της επιτρεπόμενης περιοχής τότε η κατευθυντήρια δύναμη που ασκείται είναι μηδενική και το όχημα συνεχίζει την πορεία του. Αν είναι εκτός ορίων τότε ο αλγόριθμος seek εφαρμόζεται σε κάποιο σημείο εντός της αποδεκτής περιοχής .

Η μέθοδος επιλογής του κατάλληλου σημείου θα μπορούσε να βρεθεί με τον ακόλουθο τρόπο. Η μελλοντική θέση του οχήματος προβάλεται

πάνω στο τοίχωμα που ορίζει τα σύνορα του αποδεκτού χώρου. Μετά βρίσκεται το σημετρικό σημείο ως προς το σημείο που η προβολή τέμνει την επιφάνεια του τοίχου και εφαρμόζεται ο αλγόριθμος seek για αυτό το σημετρικό σημείο το οποίο υποχρεωτικά θα βρίσκεται εντός της αποδεκτής περιοχής.



Μια άλλη μέθοδος εύρεσης τέτοιου σημείου μπορεί να είναι η εύρεση του σημείου τομής της επιφάνειας του τοίχου από την ευθεία που είναι κάθετη ως προς την τρέχουσα ταχύτητα και μετά ευρεση του σημετρικου σημειου της μελλοντικής θέσης ως προς το σημείο τομής και εφαρμογή του αλγορίθμου seek για αυτό.

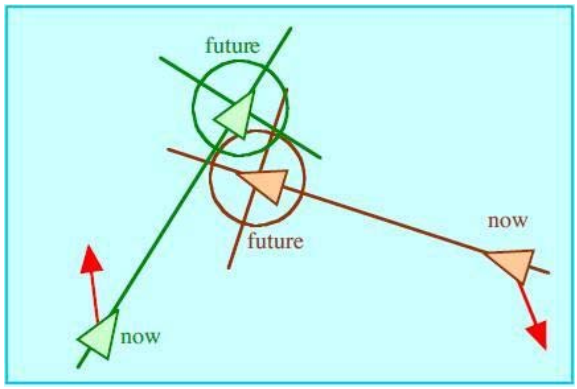
Τέλος μία ακόμη παραλλαγή containment είναι το Flow Field Following. Ένα παράδειγμα αυτου θα μπορούσε να αποτελεί ένα μαγνητικό πεδίο όπως φαίνεται από τα ρινίσματα σιδήρου όταν αυτά βρίσκονται σε μία κόλα χαρτί και ο μαγνήτης βρίσκεται από κάτω. Στην περιοχή στην οποία επιδρά μπορεί κανείς να πεί ότι η δυνάμεις που υπάρχουν μπορούν προσεγγιστικά να αναπαρσταθούν σαν βελάκια πάνω σε διάγραμμα. Ανάλογα την θέση στην οποία βρίσκεται το όχημα υπάρχει κάποιο βέλος που ορίζει την κατεύθυνση προς την οποία θα τείνει να κινείται. Το όχημα προσπαθεί να αποκτήσει ίδια διεύθυνση με το βέλος που αντιστοιχεί στην περιοχή που κινείται. Η μελλοντική θέση του οχήματος με βάση την τρέχουσα ταχύτητα ορίζει πιο βέλος θα επιδράσει στο όχημα. Το βέλος αντιπροσωπεύει την επιθυμητή ταχύτητα και η κατεύθυνση της κατευθυντήριας δύναμης είναι η διαφορά της επιθυμητής μείων της τρέχουσας ταχύτητας.

7.3.8 Unaligned Collision Avoidance

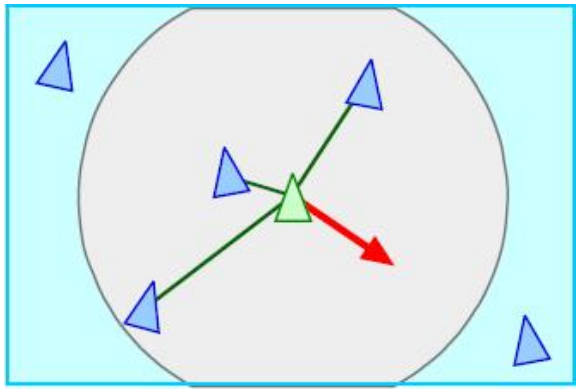
Η αποφυγή εμποδίων στην πορεία κίνησης ενός οχήματος σίγουρα αποτελεί πολύ σημαντικό τμήμα αρκετών εφαρμογών όπως παιχνίδια , βίντεο. Όμως στον πραγματικό κόσμο όταν περπατάει κανείς σε μία περιοχή η οποία έχει άλλους ανθρώπους που περπατάνε επίσης, αργά ή γρήγορα θα βρεθεί σε πορεία η οποία εκτός και μεταβληθεί θα οδηγήσει

σε σύγκρουση με κάποιον άλλο. Παρόλα αυτά είναι εφικτή η κίνηση χωρίς συγκρούσεις διότι κάθε άνθρωπος έχει την δυνατότητα να προβλέπει τόσο την δικιά του μελλοντική θέση όσο και αυτές των άλλων ανθρώπων γύρω του. Με αυτά τα στοιχεία στρίβει όσο χρειάζεται και αυξάνει η μειώνει την ταχύτητά του ώστε να αποφύγει τέτοια φαινόμενα. Αυτή είναι και η λειτουργία της Μη Ευθιγραμισμένης Αποφυγής Συγκρούσεων unaligned Collision Avoidance.

Για να επιτευχθεί το ανωτέρω αρχικά υπολογίζεται η μελλοντική θέση τόσο του οχήματος ανάφοράς όσο και των άλλων γύρω με βάση την ταχύτητα τους καθώς και πότε στο μέλλον πρόκειται πιθανώς να συγκρουστούν και επιλέγεται η σύγκρουση που πρόκειται να συμβεί συντομότερα. Αν η απόσταση των δύο είναι μεγαλύτερη από κάποια απόσταση ασφαλείας τότε η δύναμη κατεύθυνσης που προκύπτει από αυτήν την συμπεριφορά είναι μηδενική. Αντίθετα αν η απόσταση τους πρόκειται να είναι μικρότερη από την απόσταση ασφαλείας τότε και τα δύο η περισσότερα οχήματα αλλάζουν κατεύθυνση ώστε αν αποφύγουν το σημείο πρόσκρουσης κινούμενα προς αντίθετες κατευθύνσεις προς το πλάι. Επίσης μπορεί να παρουσιαστεί επιτάχυνση του ενός και επιβράδυνση του άλλου ώστε να αποφευχθεί το πέρασμα από το σημείο πρόσκρουσης την ίδια χρονική στιγμή.



7.4 Αλγόριθμοι κίνησης πλήθους

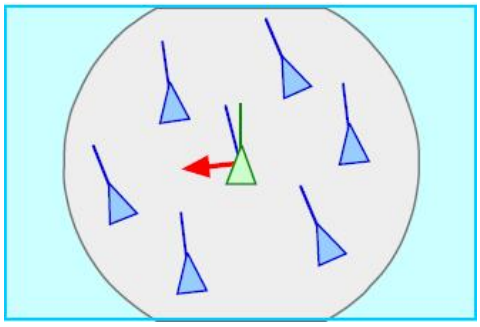


Οι τρείς συμπεριφορές που ακολουθούν έχουν να κάνουν με περισσότερα εκ των δύο οχημάτων. Ακριβέστερα έχουν να κάνουν με όλα τα οχήματα τα οποία βρίσκονται εντός κάποιας ακτίνας. Αρχικά παρουσιάζονται οι τρείς βασικές συμπεριφορές τις οποίες ο Craig Reynolds συνδύασε για να δημιουργήσει την πιο σύνθετη συμπεριφορά γνωστή και ως flocking στο boids. Παραλλαγές των συμπεριφορών θα παρουσιαστούν επίσης προς επέκταση των ήδη υπαρχόντων.

7.4.1 Separation

Η συμπεριφορά αυτή προκαλεί την διατήρηση κάποιας απόστασης από τα γειτονικά οχήματα. Πρώτα βρίσκεται η θέση όλων τον γειτονικών οχημάτων είτε ψάχνοντας όλα τα οχήματα και μετά επιλογή μόνο αυτών που είναι εντός κάποιας ακτίνας είτε κάνοντας αναζήτηση μόνο σε μια προκαθορισμένη περιοχή. Για κάθε γειτονικό όχημα υπολογίζεται μια δύναμη η οποία επιδρά στο όχημα στόχο η οποία είναι μεγαλύτερη όταν η απόσταση είναι μικρή και τείνει στο μηδέν όσο η απόσταση είναι μεγαλύτερη. Αυτό επιτυγχάνεται αν αφαιρέσουμε από την τρέχουσα θέση του οχήματος αναφοράς την θέση του οχήματος το οποίο ελέγχεται και μετά αφού κανονικοποιηθεί το διάνυσμα πολλαπλασιαστεί με κάποιον συντελεστή. Ο συντελεστής που προτείνεται από τον Craig Reynolds είναι ο 1/r. Τέλος υπολογίζεται το άθροισμα αυτών των δυνάμεων και επιδρά στο όχημα.

7.4.2 Cohesion



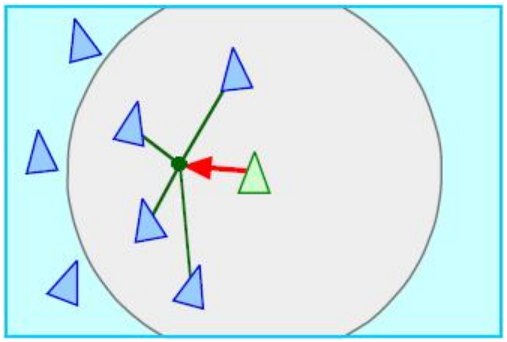
Αυτή η συμπεριφορά δίνει την δυνατότητα στο όχημα να βρίσκει και να επιδιώκει να βρεθεί στην μέση θέση σε σχέση με τα γειτονικά του οχήματα. Η μέθοδος εύρεσης των γειτονικών οχημάτων είναι η ίδια με την μέθοδο που χρησιμοποιείται στην συμπεριφορά Separation. Μετά βρίσκεται ο μέσος όρος των θέσεων των γειτονικών οχημάτων και είτε αφαιρείται η τρέχουσα θέση του οχήματος αναφοράς από το σημείο της μέσης θέσης ή μπορεί να εφαρμοστεί seek προς αυτό το σημείο.

7.4.3 Alignment

Η συμπεριφορά αυτή επιτρέπει την ευθυγράμμιση του οχήματος αναφοράς με τα οχήματα που βρίσκονται γειτονικά σε αυτό τόσο της κατεύθυνσης κίνησης όσο και τις εξίσωσης της ταχύτητας του με την μέση ταχύτητα. Αρχικά υπολογίζεται η μέση ταχύτητα όλων των γειτονικών οχημάτων η οποία γίνεται η νέα επιθυμητή ταχύτητα desired_velocity. Η δύναμη προκύπτει από την αφαίρεση της τρέχουσας ταχύτητας του οχήματος αναφοράς από την επιθυμητή ταχύτητα.

7.4.4 Flocking

Η συμπεριφορά αυτή αποτελεί την σύνθεση των προηγούμενων τριών συμπεριφορών και χρησιμοποιήθηκε για την εξομοίωση κίνησης σμηνών πουλιών ή κίνησης κοπαδιών ψαριών. Μια υλοποίηση της είναι το άθροισμα των δυνάμεων που προκύπτουν από τις παραπάνω συμπεριφορές. Μια ακόμη προσέγγιση είναι η κανονικοποίηση του διανύσματος της κάθε δύναμης και μετά πολλαπλασιασμός της κάθε μιας με κάποια συντελεστή και άθροισμα



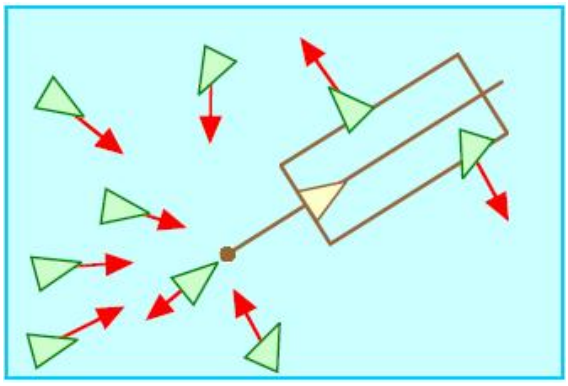
τους.

7.4.5 Εξέλιξη και η συμπεριφορά Leader Following

Εξελίσσοντας τις παραπάνω συμπεριφορές προκύπτει μια νέα συμπεριφορά κατά την οποία όλοι οι πράκτορες προσπαθούν να ακολουθήσουν έναν πράκτορα αρχηγό χωρίς να βρίσκονται στο δρόμο του σε περίπτωση που βρίσκονται μπροστά του και ακολουθώντας τον χωρίς να συνοστίζονται. Αυτό επιτυγχάνεται αν προβληθεί ένα ορθογώνιο μπροστά από τον πράκτορα αρχηγό και όσοι άλλοι πράκτορες βρεθούν μέσα σε αυτό το ορθογώνιο προβάλλονται πάνω στην γραμμή που ορίζει η κατεύθυνση κίνησης του αρχηγού. Τότε τους ασκείται δύναμη που έχει φορά που ορίζεται από την διαφορά της τρέχουσας θέσης του εκάστοτε οχήματος που βρίσκεται εντός του ορθογωνίου μείων το σημείο τομής με την γραμμή διεύθυνσης κίνησης του πράκτορα αρχηγού και μέτρο ίσο με την ακτίνα του πράκτορα αρχηγού μείων την απόσταση τους από την γραμμή διεύθυνσης κίνησης του πράκτορα αρχηγού.

Όλοι οι πράκτορες πλην του αρχηγού εφαρμόζουν arrival σε κάποιο σημείο το οποίο βρίσκεται πάνω στον άξονα κίνησης του αρχηγού αλλά πίσω από τον αρχηγό σε κάποια δεδομένη απόσταση ώστε να προσπαθούν να φτάσουν αυτό το σημείο αλλά μειώνοντας την ταχύτητά τους ώστε να γίνει ίση με του αρχηγού. Επίσης εφαρμόζουν separation για να μην συγκρούονται μεταξύ τους.

7.5 Συνδυάζοντας τις παραπάνω συμπεριφορές



Όπως παρουσιάστηκε νωρίτερα ο συνδυασμός συμπεριφορών μπορεί να δημιουργήσει πολύ πιο

ενδιαφέρουσες συμπεριφορές. Επίσης πολύ σπάνια η λύση ενός προβλήματος μπορεί να δοθεί με μία μόνο συμπεριφορά. Παραδείγματος χάρη μια αγέλη λύκων κυνηγάει κάποιο θήραμα μέσα σε ένα δάσος. Οι λύκοι προσπαθούν να φτάσουν το θήραμα ενώ ταυτόχρονα αποφεύγουν τα δέντρα και τους βράχους τους οποίους δεν μπορούν να προσπεράσουν από πάνω. Ένα άλλο πρόβλημα είναι η απόκρυψη κάποιου από ένα εμπόδιο από κάποιον που βρίσκεται στην άλλη πλευρά του εμποδίου. Το πρώτο παράδειγμα αρκεί να συνδυάσει pursuit, obstacle avoidance και flocking ή leader follow ενώ στο δεύτερο πρέπει να βρεθει το αντιδιαμετρικό σημείο ως προς τον βράχο και να εφαρμοστεί seek προς αυτό το σημείο.

Η σύνθεση συμπεριφορών μπορεί να γίνει με εφαρμογή όλων των συμπεριφορών που είναι απαραίτητες και να βρεθεί ο μέσος όρος των δυνάμεων που προκύπτουν από τις συμπεριφορές. Αυτή η μέθοδος έχει δύο βασικά μειονεκτήματα. Πρώτον απαιτεί πολύ υπολογιστική ισχύ, ποσότητα ανάλογη με τον αριθμό των συμπεριφορών καθώς επίσης και της πολυπλοκότητας τις κάθε συμπεριφοράς . Κατά δεύτερον είναι πιθανόν η δύναμη της μιας συμπεριφοράς να καταργήσει την αλλαγή που πρέπει να επέλθει εξαιτίας κάποιας άλλης. Παράδειγμα αποτελεί το περιστατικό κατά το οποίο για να αποφευχθεί ένα δέντρο ένας από τους λύκους πρέπει υποθετικά να στρίψει προς μία κατεύθυνση ενώ το θήραμα του βρίσκεται στην αντίθετη κατεύθυνση κατά συνέπεια η μία δύναμη να εξουδετερώσει την άλλη και ο λύκος να καταλήξει να συγκρουστεί με το δέντρο.

Ένας τρόπος αποφυγής του προβλήματος είναι η ανάθεση προτεραιοτήτων στην εκτέλεση των συμπεριφορών εκτελώντας μόνο μια κάθε φορά. Αυτή η μέθοδος όμως είναι πιθανόν να εκτρέψει τον λύκο από τον προορισμό του αρκετά ώστε το θήραμα να ξεφύγει. Μια άλλη δυνατότητα είναι ανάθεση συντελεστών στην δύναμη που παράγεται από την κάθε συμπεριφορά ώστε το αποτέλεσμα να ευνοεί ορισμένες συμπεριφορές στο τελικό αποτέλεσμα. Με αυτόν τον τρόπο όμως δεν αποφεύγεται η χρήση μεγάλης υπολογιστικής ισχύος.

Η μέθοδος που ο Craig Reynolds προτείνει στην μελέτη του είναι η ανάθεση πιθανοτήτων να συμβεί κάποια συμπεριφορά. Αν δεν συμβεί αυτή ελέγχεται η επόμενη και ου το καθεξής. Με αυτό τον τρόπο η πλέον σημαντικές συμπεριφορές θα έχουν μεγαλύτερη πιθανότητα να συμβούν

και οι λιγότερο σημαντικές είναι λιγότερο πιθανό να συμβούν. Με αυτόν τον τρόπο επίσης αν κάποια συμπεριφορά δεν παράγει κάποια κατευθυντήρια δύναμη ελέγχεται η επόμενη.

8 Συμπεράσματα

Στόχος αυτής της εργασίας είναι η γνωριμία τόσο με τον κόσμο του Second Life και της κατευθυντήριες συμπεριφορές καθώς επίσης και εξοικείωση με τα εργαλεία που απαιτούνται για τον προγραμματισμό διαφόρων εφαρμογών στην γλώσσα LSL.

Αρχικά γίνεται αναφορά στο τι είναι το Second Life και τι μπορεί να πραγματοποιηθεί από την πλευρά του χρήστη. Ακολουθεί η γνωριμία με τα εργαλεία Open Simulator και Imprudence Viewer και πως μπορεί να γίνει η εγκατάσταση της τρέχουσας έκδοσης του καθενός στα πλαίσια της εργασίας. Στη συνέχεια ακολουθεί αναλυτική επεξήγηση της ιδέας του προγραμματισμού μηχανών καταστάσεων και των βασικών συστατικών της scripting γλώσσας του Second Life γνωστής ως LSL.

Αναλύονται ορισμένες δυνατότητες που παρέχονται τόσο σε εικονικά αντικείμενα όσο και σε αντικείμενα που υπακούν σε φυσικούς νόμους και πώς να χρησιμοποιηθούν οι ιδιότητες φυσικής. Επίσης αναφέρεται

πώς να κατασκευαστούν και να διαχειριστούν οι πράκτορες NPCs χρησιμοποιώντας την επέκταση της γλώσσας LSL γνωστής και ως OSSL. Αμέσως μετά δίνονται οι βασικοί ορισμοί οχήματος , πιλότου και συμπεριφοράς καθώς και κάποιες παραδοχές που θα χρησιμοποιηθούν αργότερα. Τέλος αναλύονται όλες οι συμπεριφορές που ο Craig Reynolds ορίζει στην μελέτη του περί αυτόνομων πρακτόρων και γίνεται υλοποίηση μερικών από αυτές σε παραλλαγή πάντα με τον στόχο της περεταίρω ελαχιστοποίησης της υπολογιστικής ισχύος που απαιτείται για την υλοποίηση τους. Η εργασία τελειώνει με τους δυνατούς τρόπους συνδυασμών των συμπεριφορών για την ρεαλιστική αναπαράσταση διαφόρων καταστάσεων.

Οι πληροφορίες που παρέχονται σε αυτήν την μελέτη μπορούν να χρησιμοποιηθούν για την ρεαλιστικότερη και βέλτιστη αναπαράσταση καταστάσεων που μπορούν να βρεθούν στον πραγματικό κόσμο. Παράδειγμα αποτελεί πιθανώς κάποια εξομοίωση κίνησης πλήθους σε κάποιο κτίριο όταν κάποιο ατύχημα συμβαίνει όπως σεισμός ή πυρκαγιά στην οποία περίπτωση μπορεί να ελεγχθεί κατά πόσο το κτίριο αυτό τηρεί καλές προϋποθέσεις για ασφαλή έξοδο.

Ο Open Simulator όντας γραμμένος σε C# γλώσσα υποστηρίζει επεκτάσιμες εφαρμογές που θα μπορούσαν να περιέχουν τεχνητή νοημοσύνη ώστε η επιλογή της κατάλληλης συμπεριφοράς να ακολουθεί ορισμένα κριτήρια η ακόμη και συνδυασμός των συμπεριφορών για την επίλυση προβλημάτων τεχνητής νοημοσύνης.Με κατανόηση navigation meshes ή waypoints και χρήση αλγορίθμων τεχνητής νοημοσύνης θα μπορούσε να κατασκευαστεί παιχνίδι για τον κόσμο του Second Life το οποίο να εφαρμόζει τις παραπάνω συμπεριφορές.

Η μελέτη αυτή προάγει την ρεαλιστικότητα στους εικονικούς κόσμους δίνοντας μια προσέγγιση που στηρίζεται κυρίως σε φαινόμενα από την καθημερινή ζωή βοηθώντας τους χρήστες να βιώσουν σε μεγαλύτερο βαθμό έναν κόσμο που αλλιώς θα ήταν άψυχος.

Παρέχει υλικό προς σκέψη και εξέλιξη σε τομείς έρευνας και ψυχαγωγίας καθώς επίσης και εκπαίδευσης. Στον τομέα της έρευνας

παρέχει πληροφορίες απαραίτητες για την δημιουργία ενός εικονικού κόσμου ο οποίος θα μπορεί κάλλιστα να αποτελέσει την βάση για την εφαρμογή τόσο βελτιωμένων τεχνικών όσο και νέων.

Στον τομέα της ψυχαγωγίας πέρα από το ότι κάνει γνωστές τις τεχνικές προγραμματισμού της γλώσσας που χρησιμοποιείται στο Second Life και το ότι δημιουργεί την ψευδαίσθηση της τεχνητής ζωής βοηθάει παρέχοντας έναν τρόπο έκφρασης και ίσως με εξέλιξη και την δημιουργία εικονικού κόσμου για άλλους χρήστες με τους οποίους μπορεί καθένας να μοιραστεί αυτές τις εμπειρίες.

Τέλος, στον τομέα της εκπαίδευσης μπορούν να γίνουν αναπαραστάσεις διαφόρων φυσικών φαινομένων με ρεαλισμό. Φαινομένων τα οποία θα ήταν πολύ δύσκολο έως αδύνατον να τα παρατηρήσει κανείς χωρίς πολύ ακριβό εξοπλισμό. Με δημιουργία εικονικού κόσμου ανοιχτού προς όλους μπορούν να εκπονηθούν μελέτες ακόμη και της επιστήμης της κοινωνιολογίας για τις συνήθειες συγκεκριμένων ομάδων πληθυσμού στους οποίους ο εικονικός κόσμος απευθύνεται.

Θέματα που εξετάστηκαν και πηγές - Βιβλιογραφία

Πληροφορίες για το Second Life και μηχανές καταστάσεων www.wikipedia.org

Opensimulator και OSSL. Opensimulator.org

Γνωριμία με την LSL από την ιστοσελίδα του πανεπιστημίου αιγαίου σχετικά με εικονική πραγματικότητα από τον κ. Βοσινάκη Σπυρίδων. Hci-dpsd.wikispaces.com/vr

Δυνατότητες LSL και API. – wiki.secondlife.com

Maths – Transformations using Quaternions. http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/transforms/index.htm

Boids Model από Craig Reynolds. http://www.red3d.com/cwr/boids/

Steering Behaviors for autonomous characters από Craig Reynolds. http://www.red3d.com/cwr/steer/

Autonomous Steering Behaviors by Daniel Shiffman. http://www.shiffman.net/teaching/nature/steering/

Δημοσίευση στο Siggraph 2000 από τον Robin Green . Steering behaviors by Robin Green Siggraph 2000.

Δημοσίευση με βάση τις διαλέξεις του Professor Jurgen Sauer.http://fbim.fh-regensburg.de/~saj39122/feisch/Diplomarbeit/theory_eng/steering_eng.html

Παραδείγματα και υλοποίηση αλγορίθμων http://www.openprocessing.org/sketch/11045.

Navigation Meshes for FPS non Player Characters από Pierre-Marie http://racc.bots-united.com/tutorial-navmesh.html.

Τεχνικές Τεχνητής νοημοσύνης. A star αλγόριθμος και εφαρμογές για εύρεση μονοπατιού. http://www.policyalmanac.org/games/aStarTutorial.htm

Δημιουργία εφαρμογών για Second Life με χρήση C# μέσω της βιβλιοθήκης libopenmetaverse. http://lib.openmetaverse.org/wiki/Main_Page

Παράρτημα

Μαζί με την μελέτη παραδίδεται και υλοποίηση τόσο του server όσο και των υλοποιήσεων που χρησιμοποιούνται καθόλη την διάρκεια της μελέτης σε κώδικα LSL.

Τα στοιχεία για την σύνδεση είναι

First Name: Knowledge

Last Name: Lab

Password: test

Για την δημιουργία των πρακτόρων στην υλοποίηση του OpenSimulator 0.7.3 απαιτείται πρώτα η αποθήκευση εμφανίσεων με τα αντικείμενα wandervirus και

Για να γίνει αυτό πρέπει να μεταφερθεί στο κανάλι επικοινωνίας ένα το μήνυμα clone για τον πράκτορα που εκτελεί wander και clone2 για

αυτόν που εκτελεί pursuit. Επίσης κατά την αντιγραφή της εμφάνισης πρέπει να έχουμε προσαρτημένα τα αντικείμενα WanderVirus και Seek αντίστοιχα. Ο κώδικας του αντικειμένου δημιουργίας των πρακτόρων είναι ο ακόλουθος.

```
key npc=NULL_KEY;
key npc2=NULL_KEY;

default
{
   state_entry()
   {
      llListen(1,"","","");
   }

   listen(integer channel, string name, key id, string msg)
   {
      if (msg != "")
      {
         if (msg == "create" && npc == NULL_KEY)
         {

            npc = osNpcCreate("Wandering", "Disaster", <120, 157, 26>, "wanderer");
            npc2=osNpcCreate("Seeker", "of Truth", <110, 127, 26>, "seeker");


         }


         else if (msg == "clone")
         {
            osOwnerSaveAppearance("wanderer");
         }
         else if (msg == "clone2")
         {
            osOwnerSaveAppearance("seeker");
         }
         else if (msg == "destroy")
         {
            list avatars = llList2ListStrided(osGetAvatarList(), 0, -1, 3);
            integer i;
            llSay(0,"NPC Removal: No avatars will be harmed or removed in this process!");
            for (i=0; i<llGetListLength(avatars); i++)
            {
               string target = llList2String(avatars, i);
               osNpcRemove(target);
               llSay(0,"NPC Removal: Target "+target);

            }
```

```
            npc=NULL_KEY;
        }
        else
        {
            llOwnerSay("I don't understand [" + msg + "]");
        }
    }
  }
}
```

Για να εκτελέσει ο πράκτορας Wandering Disaster wander πρέπει να χρησιμοποιηθεί το δεύτερο κανάλι επικοινωνίας για να μεταφερθεί το μήνυμα wander.

Κώδικας που δεν εκτελέστηκε λόγω τεχνικών δυσκολιών.

Pursuit – Evasion

```
key npc2=NULL_KEY;
key seeker=NULL_KEY;
key target=NULL_KEY;

float desired_force_mag;

vector position;
vector velocity;
vector cur_pos;
vector cur_vel;
vector pred_pos;
vector direction;
vector desired_vel;
vector desired_force;
```

```
pursue(key seeker,key prey)
{
   list info = llGetObjectDetails(                    //eyresh theshs kai taxythtas toy Npc poy
kanoume pursuit kai anathesh se metablhtes anti gia list.Vector
                        prey,
                        [
                        OBJECT_POS,
                        OBJECT_VELOCITY
                        ]);
   position=llList2Vector(info,0);
   velocity=llList2Vector(info,1);

   pred_pos=position+ velocity*0.5;                   //h mellontikh thesh tou Npc pou
akolouthoume me bash thn trexousa thesh kai taxythta tou
   cur_pos=osNpcGetPos(seeker);                       //h trexousa thesh kai taxythta tou seeker
npc(gia thn deyterh briskoume thn taxythta tou attachment)
   cur_vel=llGetVel();

   direction=llVecNorm(pred_pos-cur_pos);             // dieythynsh pros thn opoia prepei na
kinithoume




   desired_vel=3.8*direction;                         // epithymiti taxythta (walk speed) pou tha
thelame na eixame gia na pame to syntomotero pros ton wanderer Npc

   desired_force=desired_vel-cur_vel;                 //epithymiti dynami pou prepei na askithei
(theorontas oti h maza einai 1 h dynamh kai h epitaxynsh einai to idio pragma
   desired_force_mag=llVecMag(desired_force);      //metro dynamis gia na thn
perikopsoume se kapoia timh iso me megisth epitrepomenh


   if(desired_force_mag>3)                            //perikoph an einaimegalyterh apo thn
megisth epitrepth
   {
      desired_force=desired_force/desired_force_mag;
      desired_force_mag=3;
      desired_force=desired_force*desired_force_mag;
   }


   cur_vel=cur_vel+desired_force;                     //trexoysa taxythta einai ish me thn taxythta
ayth thn stigmh syn thn dynamh (epitaxynsh)




   osNpcMoveToTarget(seeker,cur_pos+cur_vel,OS_NPC_NO_FLY);     // kinisi toy seeker
sthn thesh cur_pos + trexoysa taxythta (kanonika epi xrono alla tha perikopei apo to SL
                                                      //aytomata opote paraleipetai o
```

```
pollaplasiasmos gia exoikonomhsh isxyos


}



default
{                                                                   //katastash default sthn opoia
orizetai ena event timer gia na mhn xekinisei amesos to seek kai gia na psaxnei
                                                                    //synexos gia praktores pou den
einia o owner tou script (dld o seeker) kai na einai NPC kai oxi avatar
   state_entry()
   {

      seeker=llGetOwner();

      llSetTimerEvent(5.0);
   }

   timer()
   {
      list avatars = llList2ListStrided(osGetAvatarList(), 0, -1, 3);        //dhmiourgia ypolistas
ths listas pou prokyptei apo osGetAvatarList() poy periexei mono ta keys
      integer i;
      for (i=0; i<llGetListLength(avatars); i++)
            {
               target = llList2Key(avatars, i);                          // elegxos seiriaka ta kleidia gia
ayto pou den einai o seeker kai den einai avatar kai anathesh sthn npc2
               x=osIsNpc(target);
               if(target!=seeker && x==TRUE )
               {
                  npc2=target;                                          //otan brethei enas tote pame se
allo state afou akyrothei to timer
                  llSetTimerEvent(0.0);
                  state pursuit;

               }
            }
   }
}

state pursuit                                                       //katastash pursuit sthn opoia
kata thn eisodo ekteleitai h pursue methodos kai orizetai timer ana 0.5 sec
{
   state_entry()
   {
      pursue(seeker,npc2);
      llSetTimerEvent(0.5);

   }
```

```
  timer()                                                    // pou xanakalei thn synarthsh kathe
0.5 sec. Tha mporouse na oristei anadromika h pursuit alla isws na einai pio
{                                                            //elafry gia to systhma na trexei kathe
0.5 sec. Giayto kai to timer. Dokimasthkan kai ta dyo.
    pursue(seeker,npc2);
  }
}
```

Ανάλυση του προγράμματος

Αρχικά ορίζονται οι μεταβλητές …

```
key npc2=NULL_KEY;          //Το Key UUID του στόχου.
key seeker=NULL_KEY;        //το Key UUID του πράκτορα που εκτελεί pursuit
key target=NULL_KEY;        //Προσωρινό κλειδί για χρήση
float desired_force_mag;    //Μέτρο της επιθυμητής δύναμης
vector position;            //Τρέχουσα θέση του οχήματος στόχου
vector velocity;            //Τρέχουσα ταχύτητα του οχήματος στόχου
vector cur_pos;             //Τρέχουσα θέση του οχήματος που εκτελεί pursuit
vector cur_vel;             //Τρέχουσα ταχύτητα του οχήματος pursuit
vector pred_pos;            //Η μελλοντική θέση του οχήματος στόχου
vector direction;           //Η διεύθυνση που πρέπει να κινηθεί το όχημα pursuit
vector desired_vel;         //Η επιθυμητή ταχύτητα
vector desired_force;       //Η επιθυμητή δύναμη
```

Οι μεταβλητές ορίζονται ως global καθώς στην πράξη η δημιουργία και καταστροφή τους κατά το πέρας της μεθόδου θα κοστίζει περισσότερο σε υπολογιστική ισχύ από το να τους δοθεί μόνιμη θέση στη μνήμη εφόσον η μέθοδος θα καλείται σε πολύ μικρά χρονικά διαστήματα.

Αμέσως μετά ορίζεται η μέθοδος pursue ως εξής:

```
    pursue(key seeker,key prey)
{
  list info = llGetObjectDetails(prey,
                        [
                        OBJECT_POS,
                        OBJECT_VELOCITY
                        ]);
  position=llList2Vector(info,0);
  velocity=llList2Vector(info,1);

  pred_pos=position+ velocity*0.5;
cur_pos=osNpcGetPos(seeker);
 cur_vel=llGetVel();

  direction=llVecNorm(pred_pos-cur_pos );
```

```
   desired_vel=3.8*direction;

   desired_force=desired_vel-cur_vel;
desired_force_mag=llVecMag(desired_force);

   if(desired_force_mag>3)
   {
      desired_force=desired_force/desired_force_mag;
      desired_force_mag=3;
      desired_force=desired_force*desired_force_mag;
   }


   cur_vel=cur_vel+desired_force;



   osNpcMoveToTarget(seeker,cur_pos+cur_vel,OS_NPC_NO_FLY);

}
```

Αρχικά ανατίθενται τα χαρακτηριστικά θέση και ταχύτητα του στόχου στις μεταβλητές position και velocity τα οποία συλλέγονται μέσω της llGetObjectDetails σε μια λίστα και μετά με την llList2Vector καταλήγουν ως τιμές στις μεταβλητές.

```
      list info = llGetObjectDetails(prey,
            [
                  OBJECT_POS,
                  OBJECT_VELOCITY
                  ]);
   position=llList2Vector(info,0);
   velocity=llList2Vector(info,1);
```

Εκεί βρίσκεται η μελλοντική θέση του στόχου και αφού ευρεθούν και ανατεθούν οι τιμές της τρέχουσας θέσης και ταχύτητας του πράκτορα αναφοράς που εκτελεί pursuit , υπολογίζεται η νέα διεύθυνση στην οποία πρέπει ο πράκτορας αυτός να κινηθεί. Θεωρώντας ότι η μέθοδος θα εκτελείται κάθε 0.5 δευτερόλεπτα ο υπολογισμός της μελλοντικής θέσης θα προκύπτει από το άθροισμα της θέσης του οχήματος στόχου και του γινομένου της ταχύτητας του επί την χρονική μονάδα ανά την οποία θα εκτελείται η μέθοδος δηλαδή 0.5 δευτερόλεπτα στην συγκεκριμένη περίπτωση. Έτσι…

```
pred_pos=position+ velocity*0.5;
cur_pos=osNpcGetPos(seeker);
 cur_vel=llGetVel();
 direction=llVecNorm(pred_pos-cur_pos );
```

Αμέσως μετά ευρίσκεται η επιθυμητή ταχύτητα και η δύναμη που απαιτείται ώστε ο πράκτορας να την αποκτήσει. Κατά τον υπολογισμό θεωρείται ότι δύναμη και επιτάχυνση είναι ίδια εφόσον κατά τον σχεδιασμό θεωρείται ότι ο πράκτορας μπορεί να αναπαρασταθεί ρεαλιστικά ως σημειακό όχημα με κάποιες παραλλαγές . Κατά συνέπεια η μάζα του για το πρόγραμμα είναι ίση με μία μονάδα. Επίσης υπολογίζεται και το μέτρο της επιθυμητής δύναμης. Η ταχύτητα προκύπτει αν το κανονικοποιημένο διάνυσμα που ορίζει την δίευθυνση κίνησης πολλαπλασιαστεί με την μέγιστη ταχύτητα βαδίσματος (αφού θέλουμε ο πράκτορας να πηγαίνει στον προορισμό του βαδίζοντας).

```
desired_vel=3.8*direction;
desired_force=desired_vel-cur_vel;
desired_force_mag=llVecMag(desired_force);
```

Όμως η μέγιστη δύναμη εξαρτάται από το πόσο δυνατός είναι ο πράκτορας. Κατά συνέπεια είναι προκαθορισμένη και αν η απόκτηση της επιθυμητής ταχύτητας απαιτεί μεγαλύτερη δύναμη τότε αυτή η δύναμη θα πρέπει να περικοπεί ώστε να μην ξεπερνάει την μέγιστη δύναμη. Έτσι αν η δύναμη είναι μεγαλύτερη από τρία (μέγιστη επιτρεπτή) τότε γίνεται κανονικοποίηση του διανύσματος επιθυμητής δύναμης για να διατηρηθεί η διεύθυνση και μετά αφού η τιμή του μέτρου οριστεί σε 3 (το μέγιστο), με πολλαπλασιασμό του νέου μέτρου με την διεύθυνση δίνει την νέα δύναμη.

```
if(desired_force_mag>3)
                                                                    {
                                    desired_force=desired_force/desired_force_mag;
                                                        desired_force_mag=3;
                                    desired_force=desired_force*desired_force_mag;
                                                                    }
```

Η ταχύτητα του πράκτορα που εκτελεί pursuit αλλάζει στο άθροισμα της τρέχουσας και του διανύσματος που προέκυψε μετά τον έλεγχο αν η επιθυμητή δύναμη είναι μεγαλύτερη σε μέτρο από την επιτρεπτή.

```
cur_vel=cur_vel+desired_force;
```

Τέλος ορίζεται στον πράκτορα η θέση στην οποία πρέπει να κινηθεί περπατώντας (γιαυτό και OS_NPC_NO_FLY), η οποία θα ισούται με την τρέχουσα θέση του συν το γινόμενο της ταχύτητας που προέκυψε επί τον χρόνο ανά τον οποίο αναμένεται να εκτελείται η μέθοδος.

```
osNpcMoveToTarget(seeker,cur_pos+cur_vel*0.5,OS_NPC_NO_FLY);
```

Επίσης ορίζονται δύο καταστάσεις στο πρόγραμμα. Η default στην οποία πραγματοποιείται κάθε πέντε δευτερόλεπτα ένας έλεγχος στις

ενσαρκώσεις στην περιοχή και μετά με σειριακό έλεγχο όταν βρεθεί μια τιμή κλειδιού η οποία είναι διαφορετική από αυτήν που κατέχει ο πράκτορας και είναι τιμή NPC τότε ορίζεται αυτή η τιμή ορίζεται ως ο στόχος και το σύστημα μεταβαίνει στην κατάσταση pursuit αφού πρώτα μηδενιστεί το συμβάν timer.

```
default
{
   state_entry()
   {
      seeker=llGetOwner();

      llSetTimerEvent(5.0);
   }

   timer()
   {
      list avatars = llList2ListStrided(osGetAvatarList(), 0, -1, 3);
      integer i;
      for (i=0; i<llGetListLength(avatars); i++)
            {
               target = llList2Key(avatars, i);
               if(target!=seeker && osIsNpc(target))
               {
                  npc2=target;
                  llSetTimerEvent(0.0);
                  state pursuit;

               }
            }
   }
}
```

Η δεύτερη κατάσταση είναι η pursuit η οποία κατά την είσοδο σε αυτήν ορίζει να εκτελεστεί η μέθοδος pursuit με ορίσματα το id του πράκτορα που έχει το script, ο οποίος θα προσπαθεί να φτάσει τον πράκτορα του οποίου το key uuid βρέθηκε στην default κατάσταση και το οποίο θα αποτελεί το δεύτερο όρισμα. Η κλήση της μεθόδου επαναλαμβάνεται κάθε 0.5 δευτερόλεπτα όπως ορίζεται σε timer συμβάν σε αυτήν την κατάσταση. Θα μπορούσε να είχε οριστεί να εκτελείται αναδρομικά η μέθοδος pursuit αλλά επιλέχθηκε ο τρόπος αυτός για μείωση των απαιτήσεων σε επιδόσεις από το συστημα.

```
state pursuit
{
   state_entry()
   {
      pursue(seeker,npc2);
```

```
    llSetTimerEvent(0.5);

  }
  timer()
  {
    pursue(seeker,npc2);
  }
}
```

Επίσης έγινε μία προσπάθεια να χρησιμοποιηθεί και μεταβλητή ταχύτητα του πράκτορα που εκτελεί pursuit έτσι ώστε να διατηρεί συγκεκριμένη απόσταση από τον πράκτορα που έκτελεί offset pursuit, υπολογίζοντας την ταχύτητά του με συντελεστή κάποια απόσταση ασφαλείας. Το πρόβλημα τόσο με αυτή την προσεγγίση όσο και με την παραπάνω είναι ότι απαιτείται πολύς χρόνος και υπολογιστική ισχύς για το μηχάνημα δοκιμών οπότε παρουσιάζεται στο παράρτημα ως κώδικας που δεν έτρεξε.

Ο κώδικας παρατίθεται στη συνέχεια.

```
key npc2=NULL_KEY;
key seeker=NULL_KEY;
key target=NULL_KEY;

float safe_dist=10.0;
float speed_factor;
float speed;
float distance;
float max_speed=5.7;
float desired_force_mag;
float cur_vel_mod;


vector position;
vector velocity;
vector cur_pos;
vector cur_vel;
vector pred_pos;
vector direction;
vector desired_vel;
vector desired_force;

pursue(key seeker,key prey)
{
  list info = llGetObjectDetails(
```

```
                    prey,
                    [
                    OBJECT_POS,
                    OBJECT_VELOCITY
                    ]);
position=llList2Vector(info,0);
velocity=llList2Vector(info,1);

pred_pos=position+ velocity*0.5;
cur_pos=osNpcGetPos(seeker);
cur_vel=llGetVel();

direction=llVecNorm(pred_pos-cur_pos);
distance=llVecMag(pred_pos-cur_pos);

speed_factor=1+0.5*(distance/(safe_dist-1.5));
speed=speed_factor*3.8;

if(speed<max_speed)
{
desired_vel=direction*speed;
}
else
{
   desired_vel=direction*max_speed;
}

desired_force=desired_vel-cur_vel;
desired_force_mag=llVecMag(desired_force);
desired_force=desired_force/desired_force_mag;

if(desired_force_mag>3)
{
   desired_force_mag=3;
}

desired_force=desired_force*desired_force_mag;

cur_vel=cur_vel+desired_force;

cur_vel_mod=llVecMag(cur_vel)/3.8;

if(cur_vel_mod<speed_factor)
{

   osSetSpeed(seeker,cur_vel_mod);

}
else
{
   osSetSpeed(seeker,speed_factor);
```

```
    }
    osNpcMoveToTarget(seeker,cur_pos+cur_vel,OS_NPC_NO_FLY);


}



default
{
    state_entry()
    {
        seeker=llGetOwner();

        llSetTimerEvent(5.0);
    }

    timer()
    {
        list avatars = llList2ListStrided(osGetAvatarList(), 0, -1, 3);
        integer i;
        for (i=0; i<llGetListLength(avatars); i++)
            {
                target = llList2Key(avatars, i);
                if(target!=seeker && osIsNpc(target))
                {
                    npc2=target;
                    llSetTimerEvent(0.0);
                    state pursuit;

                }
            }
    }
}

state pursuit
{
    state_entry()
    {
        pursue(seeker,npc2);
        llSetTimerEvent(0.5);

    }
    timer()
    {
        pursue(seeker,npc2);
    }
}
```