ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ
ΥΠΟΛΟΓΙΣΤΩΝ
ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Διαπροσωπείες επαγωγής συναισθημάτων στην επικοινωνία ανθρώπου-μηχανής

Διπλωματική Εργασία

Σύρρου Αθηνά

Επιβλέπων : Στέφανος Κόλλιας
Καθηγητής Ε.Μ.Π

Αθήνα, Οκτώβριος 2007

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Διαπροσωπείες επαγωγής συναισθημάτων στην επικοινωνία ανθρώπου-μηχανής

Διπλωματική Εργασία
Σύρρου Αθηνά

Επιβλέπων : Στέφανος Κόλλιας
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την .....Οκτωβρίου 2007

.....	.....	.....
Στέφανος Κόλλιας	Ανδρέας Σταφυλοπάτης	Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.	*Καθηγητής Ε.Μ.Π.*	*Καθηγητής Ε.Μ.Π.*

Αθήνα, Οκτώβριος 2007

Περίληψη

Η παρούσα αναφορά περιγράφει την εφαρμογή (ηλεκτρονικό παιχνίδι) που αναπτύχθηκε με τη βοήθεια μίας πλατφόρμας που επιτρέπει την ανάπτυξη αλληλεπιδραστικών εφαρμογών, προκειμένου να μελετηθεί η συναισθηματική αλληλεπίδραση ανθρώπου-μηχανής. Σκοπός της εφαρμογής είναι η πρόκληση και η καταγραφή των συναισθημάτων του χρήστη ο οποίος καλείται να αντιμετωπίσει απρόσμενα γεγονότα κατά την εξέλιξη της εφαρμογής. Η καταγραφή των συναισθημάτων μπορεί να βοηθήσει στη δημιουργία μίας βάσης δεδομένων και εν συνεχεία ενός έμπειρου συστήματος που θα μπορεί να αναγνωρίζει δυναμικά τη συναισθηματική κατάσταση του χρήστη. Συνοπτικά η διάρθρωση της αναφοράς είναι η ακόλουθη:

- Το **1ο Κεφάλαιο** αποτελεί μία συνολική επισκόπιση του αντικειμένου που πραγματεύεται η συγκεκριμένη εργασία.
- Στο **2ο Κεφάλαιο** πραγματοποιείται μία επισκόπιση του ερευνητικού χώρου επικοινωνίας ανθρώπου-μηχανής (συναισθηματική υπολογιστική) και συγκεκριμένα αναλύεται η συναισθηματική αλληλεπίδραση ανθρώπου-μηχανής καθώς και οι διαδικασίες πρόκλησης, καταγραφής και ανάλυσης συναισθημάτων με τη βοήθεια της χρήσης ηλεκτρονικών παιχνιδιών.
- Το **3ο Κεφάλαιο** αναφέρεται στη μηχανή του Torque με τη βοήθεια της οποίας υλοποιήθηκε η συγκεκριμένη εφαρμογή.
- Στο **4ο Κεφάλαιο** δίνεται αναλυτικά η αρχιτεκτονική της εφαρμογής που υλοποιήθηκε και περιγράφεται στην αναφορά.
- Στο **5ο Κεφάλαιο** περιγράφονται και παρουσιάζονται οι τρόποι παρέμβασης που επιλέχθηκαν και υλοποιήθηκαν προκειμένου να συλλέξουμε τα δεδομένα που αφορούν τη συναισθηματική κατάσταση του χρήστη και τις αντιδράσεις του που μεταβάλλονται ανάλογα με αυτήν.
- Το **6ο Κεφάλαιο** περιλαμβάνει αξιολόγηση της μεθόδου που εφαρμόστηκε, συμπεράσματα που μπορούν να εξαχθούν καθώς και πιθανές μελλοντικές εξελίξεις.
- Ακολουθεί **παράρτημα** όπου παρατίθεται αναλυτικά ο κώδικας της εφαρμογής και τέλος η αναφορά στη **βιβλιογραφία** που χρησιμοποιήθηκε.

Abstract

This essay describes the application (computer game) which was developed using a platform that allows us to develop interactive applications, in order to study affective computing issues regarding emotion induction through a gaming interface. So, the purpose of this application is to induce user's emotions and monitor their evolution. The emotions will be challenged through some unexcpected events that will take place while the user is playing a computer game that has been developed for this purpose. The monitoring of the emotions will help us develop a database and then an expert system which will be able to dynamically recognize the emotional state of a user.
The structure of this essay is the following:

- The **1st Chapter** is an overall review of the main theme of this essay.
- The **2nd Chapter** deals with the meaning of 'affective computing' and more specifically with the emotional interaction between human and computer. Furthermore, in this chapter we mention the ways of inducing, monitoring and analyzing the emotions using digital applicatons (f.e. computer games)
- The **3rd Chapter** refers to the game engine platform (Torque) that was used in order for this application to be developed.
- In the **4th Chapter** the architecture of this application is described.
- The **5th Chapter** refers to the ways of intervention that we decided to put into practice and the way we finally did it in order to collect data that have to do with the user's emotional state.
- In the **6th Chapter** there is a valuation of the method that our application uses and there are also some conclusions and possible future evolution.
- In the **Appendix** we have included the code of the application which is followed by the **bibliography** that was used for this report.

Λέξεις-Κλειδιά

- **Affetcive computing:** Συναισθηματική υπολογιστική. Ουσιαστικά η συναισθηματική αλληλεπίδραση ανθρώπου-μηχανής.
- **Arousal:** Μία από τις δύο συνιστώσες που περιγράφουν την έννοια του συναισθήματος. Αναφέρεται στο επίπεδο του ενθουσιασμού ή διαφορετικά στην έντασή του.
- **DataBlock:** Ειδικός τύπος αντικειμένου που περιέχει ένα σύνολο χαρακτηριστικών που χρησιμοποιούνται για να περιγράψουν τις ιδιότητες ενός άλλου αντικειμένου.
- **Dedicated server:** Μορφή εξυπηρετητή (server) χωρίς γραφικό περιβάλλον.
- **GUI (Graphic User Interface):** Το γραφικό περιβάλλον.
- **Handle:** Κατά μία έννοια το handle είναι όμοιο με το ID. Τα πάντα στην εφαρμογή έχουν ένα handle μέσω του οποίου μπορούν να αναγνωριστούν.
- **ID:** Ο αριθμός τον οποίο δίνει η εφαρμογή σε κάθε αντικείμενο. Συνήθως τα διαφορετικά αντικείμενα αναγνωρίζονται μέσω του ID τους.
- **Interactive:** Αλληλεπιδραστικό χαρακτηρίζεται ένα αντικείμενο το οποίο αντιδρά σε ενέργειες του χρήστη.
- **Interface:** Το γραφικό περιβάλλον.
- **NetEvent:** Δικτυακό γεγονός. Μία διαδικασία που μεταφέρει γεγονότα στα μέλη μίας δικτυακής κοινότητας.
- **Physiological:** Αυτός που έχει σχέση με τη φυσιολογία του ατόμου.
- **Psychophysiological:** Αυτός που έχει σχέση με την ψυχολογία και παράλληλα με τη φυσιολογία του ατόμου.
- **Texture:** Υφή που μπορεί να έχει ένα αντικείμενο, δάπεδο κτλ. Καποιες φορές υποδηλώνει και το υλικό κατασκευής.
- **Torque:** Η μηχανή η οποία χρησιμοποιείται για να τρέχει την παρούσα εφαρμογή. Δημιουργία της GarageGames.
- **TorqueScript:** Το κομμάτι κώδικα της μηχανής Torque. Είναι η γλώσσα που χρησιμοποιείται σε όλη την εφαρμογή.
- **Valence:** Η δεύτερη συνιστώσα της έννοιας του συναισθήματος που αφορά τη χροιά του.

Περιεχόμενα

1 ΕΙΣΑΓΩΓΗ

Στην παρούσα εργασία χρησιμοποιήθηκε μία πλατφόρμα που μας επιτρέπει ανάπτυξη αλληλεπιδραστικών εφαρμογών, προκειμένου να μελετηθούν οι αλλαγές που σημειώνονται στη φυσιογνωμία ενός ατόμου όταν συμβαίνουν γεγονότα που επιδρούν στα συναισθήματά του. Τα γεγονότα αυτά λαμβάνουν χώρα κατά τη διάρκεια της εφαρμογής (ηλεκτρονικό παιχνίδι) κατά τρόπο απρόσμενο για το χρήστη. Η ιδέα για τη χρήση ενός ηλεκτρονικού παιχνιδιού για το σκοπό αυτό, είχε προταθεί από το Scherer όπου σε εργασία του το 1984 είχε προβάλλει ως δύο βασικά χαρακτηριστικά των ηλεκτρονικών παιχνιδιών που τα καθιστούν κατάλληλα για τη συγκεκριμένη χρήση, την εγγενή ευχαρίστηση που προκαλούν και την ύπαρξη ενός στόχου που προκαλεί επιθυμία για εξέλιξη. Αυτά τα δύο χαρακτηριστικά κρατούν το ενδιαφέρον και τα αντανακλαστικά του χρήστη σε εγρήγορση και συνεπώς είναι πιο πιθανόν να αντιδράσει με τον τρόπο που αναμένουμε σε κάποιο εξωτερικό ερέθισμα που θα προκληθεί. Φυσικά, υπάρχει πάντα ο αστάθμητος παράγοντας της πρότερης συναισθηματικής κατάστασης του ατόμου που θα πρέπει να λαμβάνεται υπόψιν στα πειράματα.

Η συγκεκριμένη διαδικασία παρέχει κάποια ουσιώδη πλεονεκτήματα σε σχέση με τις συμβατικές μεθόδους τα οποία συνοψίζονται στα παρακάτω:

- Στις μέχρι τώρα εφαρμογές, όποτε χρειαζόταν να μελετηθεί ή να καταγραφεί η απεικόνιση κάποιου συναισθήματος στο πρόσωπο ενός χρήστη, η διαδικασία ήταν επιτειδευμένη, έλειπε ο αυθορμητισμός και συνεπώς το αποτέλεσμα δεν ήταν το επιθυμητό. Η εφαρμογή που αναπτύξαμε, δίνει τη δυνατότητα για πρόκληση αυθόρμητων συναισθημάτων και εκφράσεων μέσω απρόσμενων γεγονότων που λαμβάνουν χώρα και τα οποία εντοπίζονται με τη βοήθεια ηχητικών markers καθώς, κάθε γεγονός συνοδεύεται από κατάλληλο ήχο που καταγράφεται από την κάμερα και μπορεί να εντοπιστεί.
- Συνάμα, η καταγραφή που γινόταν στα ως τώρα πειράματα ήταν στιγμιαία και δεν μπορούσε να μελετηθεί η εξέλιξη στο χρόνο. Εφόσον στην περίπτωσή μας η καταγραφή είναι συνεχής, μπορούμε να μελετήσουμε την πλήρη εξέλιξη της έκφρασης του συναισθήματος στο πρόσωπο του χρήστη.
- Ένα ακόμα πλεονέκτημα της συγκεκριμένης διαδικασίας είναι ότι στα πλαίσια ενός ηλεκτρονικού παιχνιδιού ο στόχος του χρήστη είναι ξεκάθαρος και αποβλέπει στη νίκη, συνεπώς μπορούμε να προβλέψουμε με αρκετή σιγουριά την αντίδρασή του σε εξωτερικά ερεθίσματα που θα του προκληθούν. Εάν δηλαδή το ερέθισμα δυσχεραίνει την πορεία προς τη νίκη, η αναμενόμενη αντίδραση είναι ένα αρνητικό συναίσθημα (δυσαρέσκεια, αγανάκτιση, θυμός) ενώ σε διαφορετική περίπτωση όπου το ερέθισμα επιταχύνει τη διαδικασία της νίκης, το συναίσθημα θα έχει θετική χροιά. Αντίθετα, η πρόκληση συναισθημάτων υπό οποιεσδήποτε άλλες συνθήκες είναι σαφώς πιο πολύπλοκη γιατί οι στόχοι του ατόμου δεν είναι ξεκάθαροι και συνεπώς οι αντιδράσεις μπορεί να είναι επιτειδευμένες.

Στη συνέχεια της αναφοράς θα περιγράψουμε την ακριβή διαδικασία που ακολουθήθηκε για την ανάπτυξη της συγκεκριμένης εφαρμογής και θα αναφερθούμε εκετενέστερα στο επιστημονικό πεδίο που αφορά την αλληλεπίδραση ανθρώπου – μηχανής καθώς και στη διαδικασία πρόκλησης συναισθημάτων.

2 ΣΥΝΑΙΣΘΗΜΑΤΙΚΗ ΥΠΟΛΟΓΙΣΤΙΚΗ

2.1 Εισαγωγή

Η συναισθηματική υπολογιστική έχει περιγραφεί ως 'υπολογιστική που σχετίζεται ή επηρεάζει τα συναισθήματα ή προκύπτει από αυτά'. Είναι εύλογο όμως να γεννηθεί το ερώτημα γιατί να δημιουργήσουμε έναν συναισθηματικό-αλληλεπιδραστικό υπολογιστή. Η απάντηση είναι απλή: Στις μέρες μας, τα υπολογιστικά συστήματα αλληλεπιδρούν με τους χρήστες με τρόπους που δεν επιτρέπουν περιπλοκές που προκύπτουν από την φυσιοκρατική κοινωνική αλληλεπίδραση. Ακόμη, πρόσφατες ενδείξεις δείχνουν ότι οι άνθρωποι έχουν την έμφυτη τάση να ανταποκρίνονται στα υπολογιστικά μέσα με φυσικούς και κοινωνικούς τρόπους που καθρεφτίζουν τους τρόπους με τους οποίους επικοινωνούν μεταξύ τους στα πλαίσια των κοινωνικών συναναστροφών. Τα υπολογιστικά συστήματα μπορούν να βασιστούν σε αυτή την προδιάθεση προσφέροντας στους χρήστες αλληλεπίδραση εμπλουτισμένη με συναισθηματική κατανόηση. Τα σύγχρονα συστήματα δε διαθέτουν ακόμα επιλογές που να αφορούν την κατανόηση της επικοινωνίας από το χρήστη και την κατανόηση του ίδιου του χρήστη. Ένας υπολογιστής που θα είχε τη δυνατότητα να αποκωδικοποιήσει και να παράγει συναισθηματικές αντιδράσεις θα μπορούσε να βελτιώσει ουσιαστικά τις σημερινές αλληλεπιδραστικές του δυνατότητες.

Ήδη τα πεδία της σύνθεσης και ανάλυσης συναισθημάτων αρχίζουν να ανοίγουν νέους ορίζοντες στους ερευνητές για να σκεφτούν και να δημιουργήσουν αλληλεπιδραστικά υπολογιστικά συστήματα. Ορισμένοι έχουν αρχίσει να δημιουργούν υπολογιστικά συστήματα τα οποία μπορούν να συνθέσουν συναισθηματικές καταστάσεις ενώ κάποιοι άλλοι έχουν εστιάσει το ενδιαφέρον τους στη δημιουργία τεχνικών αναγνώρισης από ακουστικά, οπτικά ή και άλλου είδους δεδομένα. Σε κάθε περίπτωση, το σχέδιο και η δομή των υπολογιστικών συστημάτων που προσπαθούν να εξάγουν συμπεράσματα της συναισθηματικής κατάστασης ενός χρήστη, πρέπει να λάβουν υπόψιν πολλές και ταυτόχρονες μεταβλητές κατάστασης. Πριν οι ερευνητές μπορέσουν να ορίσουν επαρκώς τις μικρολεπτομέρειες σε μία δοσμένη συναισθηματική ανταλλαγή, πρέπει να καταλάβουμε πώς ακριβώς θα συμπεράνουμε την απόκριση του χρήστη μέσα από το περιβάλλον του. Μόνο τότε θα μπορέσουμε να δώσουμε τη δυνατότητα στους υπολογιστές να ερμηνεύσουν τη συναισθηματική απόκριση καθώς είναι ενσωματωμένη στις πράξεις και τις εμπειρίες του ατόμου και εξαρτάται άμεσα από αυτές.

2.2 Πρόκληση και καταγραφή συναισθημάτων με την χρήση ηλεκτρονικών παιχνιδιών

Έχει γίνει ήδη αναφορά στην έννοια του συναισθήματος και σε αυτό το σημείο θα ήταν χρήσιμο να αποσαφηνίσουμε αυτό τον όρο και να ορίσουμε πώς περιγράφεται το συναίσθημα στον επιστημονικό κόσμο. Πολλοί ερευνητές υιοθετούν τη θεωρία της Whissel και περιγράφουν το συναίσθημα με ένα σύνολο δύο ή περισσότερων διαστάσεων. Οι πιο συνηθισμένες δύο διαστάσεις για την περιγραφή του συναισθήματος είναι η ***arousal*** (ενεργοποίηση του επιπέδου ενθουσιασμού -ένταση) και ***valence*** (η θετική ή αρνητική χροιά του συναισθήματος). Για να γίνουν περισσότερο κατανοητές αυτές οι έννοιες αναφέρουμε κάποια παραδείγματα: ο θυμός και ο φόβος ανήκουν στην κατηγορία 'υψηλό arousal και αρνητικό valence', η χαρά ανήκει στην κατηγορία 'υψηλό

arousal και θετικό valence’ και η λύπη ανήκει στην κατηγορία ‘χαμηλό arousal και αρνητικό valence’.

Πέρα όμως από τη συγκεκριμένη προσέγγιση, πολλοί άλλοι ερευνητές ασχολήθηκαν με τη θεωρία των συναισθημάτων και διατύπωσαν τις θεωρίες τους. Ανάμεσά τους ξεχωρίζουμε τον Plutchik ο οποίος στην ψυχολογική του θεωρία για τα συναισθήματα διατύπωσε δέκα βασικά αξιώματα:

1. Η έννοια του συναισθήματος ισχύει σε όλα τα εξελικτικά επίπεδα σε περιπτώσεις ανθρώπων, αλλά και ζώων.
2. Τα συναισθήματα παρουσιάζουν μία εξελικτική πορεία και δημιουργούν ποικιλία εκφράσεων.
3. Τα συναισθήματα βοήθησαν τους διάφορους οργανισμούς να αντιμετωπίσουν ζητήματα επιβίωσης που δημιουργήθηκαν από το περιβάλλον.
4. Παρά τις διαφορετικές μορφές έκφρασης των συναισθημάτων στα διαφορετικά είδη, υπάρχουν ορισμένα κοινά στοιχεία ή πρότυπα προσδιορίσιμα.
5. Υπάρχει ένας μικρός αριθμός βασικών συναισθημάτων.
6. Όλα τα άλλα συναισθήματα είναι μίξεις καταστάσεων ή παράγωγες περιπτώσεις, δηλαδή προκύπτουν από τα βασικά συναισθήματα ή από συνδυασμό αυτών.
7. Τα βασικά συναισθήματα είναι υποθετικά κατασκευάσματα ή ιδανικές περιπτώσεις, των οποίων οι ιδιότητες και τα χαρακτηριστικά μπορούν να προκύψουν από διαφορετικά είδη.
8. Τα βασικά συναισθήματα μπορούν να θεωρηθούν ως αντίθετοι πόλοι συγκεκριμένων ζευγών.
9. Όλα τα συναισθήματα διαφοροποιούνται στο βαθμό ομοιότητας του ενός με το άλλο.
10. Κάθε συναίσθημα μπορεί να υπάρξει σε διαφορετικούς βαθμούς έντασης και με διαφορετικά επίπεδα διέγερσης.

Σχεδιάσαμε λοιπόν ένα πείραμα με το οποίο ερευνήσαμε τις μεταβολές της συναισθηματικής κατάστασης του χρήστη όπως αυτές αποτυπώνονται στις εκφράσεις του προσώπου του, ανάλογα με τα διάφορα ερεθίσματα που δέχεται από το περιβάλλον. Τα ερεθίσματα αυτά είναι απρόβλεπτα και προκαλούνται από ένα ηλεκτρονικό παιχνίδι που καλείται ο χρήστης να παίξει χωρίς ο ίδιος να γνωρίζει την πραγματική φύση του πειράματος. Η συγκεκριμένη διαδικασία μας επιτρέπει:

- Να παρακολουθήσουμε την ολοκληρωμένη εξέλιξη της συναισθηματικής κατάστασης του χρήστη μέσω της αντιστοίχησής της σε συγκεκριμένες εκφράσεις του προσώπου του.
- Να προκαλέσουμε συναισθήματα (emotion induction) μέσα σε ένα καλά καθορισμένο περιβάλλον γεγονός που μας επιτρέπει να μπορούμε να προβλέψουμε τις πιθανές αντιδράσεις με μικρή πιθανότητα σφάλματος, έχοντας ταυτόχρονα μεγάλη σιγουριά για την αυθεντικότητά τους.

Το αλληλεπιδραστικό αυτό περιβάλλον είναι ουσιαστικά ένα ηλεκτρονικό παιχνίδι, που αποτελεί σε μεγάλο βαθμό και βασικό αντικείμενο της συγκεκριμένης αναφοράς και θα αναλυθεί διεξοδικά στη συνέχεια αλλά για διευκόλυνση του αναγνώστη η βασική του δομή θα περιγραφεί συνοπτικά σε αυτό το σημείο: Πρόκειται για μία εφαρμογή σχεδιασμένη με την πλατφόρμα Torque Game Engine (TGE) η οποία

επιτρέπει την δημιουργία εικονικών διαδραστικών κόσμων. Τον βασικό άξονα αποτελεί ο χρήστης ο οποίος οδηγεί ένα μηχανοκίνητο όχημα και προκειμένου να νικήσει θα πρέπει να φτάσει σε σκορ δέκα, με τρόπο που θα αναλυθεί στα επόμενα. Στο δρόμο του παρεμβάλλονται εμπόδια με τα οποία αν συγκρουστεί χάνει το ένα πέμπτο της ζωής του. Το παιχνίδι ελέγχεται από έναν εξυπηρετητή που δεν είναι ορατός από τον παίχτη και σκοπός του είναι να του προκαλεί απρόσμενα γεγονότα σε τακτά χρονικά διαστήματα. Τα απρόσμενα αυτά γεγονότα θα πρέπει να είναι ικανά να προκαλέσουν συναισθήματα όπως έκπληξη, χαρά, θυμό, φόβο, απογοήτευση και θα μελετηθούν με τη βοήθεια των εκφράσεων του προσώπου του χρήστη καθώς κατά τη διάρκεια διεξαγωγής του πειράματος η διαδικασία θα καταγράφεται με τη βοήθεια κάμερας.

Συνολικά, θα μπορούσαμε να πούμε ότι υπάρχουν πολλές μέθοδοι που μπορούν να δώσουν σε έναν υπολογιστή πρόσβαση σε διάφορες πλευρές της συναισθηματικής ανταπόκρισης. Προφανείς επιλογές είναι η οπτική και ακουστική ανάλυση για την αναγνώριση των εκφράσεων του προσώπου, των χειρονομιών και της φωνής, εφόσον μάλιστα μεσολαβεί απόσταση στην επικοινωνία. Η απόκριση της φυσιολογίας του ατόμου είναι πιο δύσκολα κατανοητή και ίσως απαιτεί φυσική επαφή μεταξύ του χρήστη και του υπολογιστικού συστήματος για να γίνει κατανοητή.

Επιπρόσθετα, η συλλογή πληροφοριών από την επιφάνεια του δέρματος (βιοσήματα) μπορεί να γίνει ανεπιθύμητα περίπλοκη. Οι περισσότεροι σύγχρονοι αισθητήρες βιολογικών σημάτων έχουν μία διεπιφάνεια και καλώδια που μπορούν να γίνουν ενοχλητικά. Γι' αυτό το λόγο, οι μετρήσεις που έχουν να κάνουν με τη φυσιολογία ανατίθενται βαθμιαία σε συσκευές με τις οποίες οι άνθρωποι έχουν φυσική επαφή. Παρόλο που οι αισθητήρες που χρησιμοποιήθηκαν στα αρχικά πειράματα και περιγράφονται παρακάτω ήταν ιατρικοί αισθητήρες που τοποθετούνταν στο χέρι, αυτοί οι ίδιοι αισθητήρες έχουν πλέον ενσωματωθεί μέσα σε διάφορα καθημερινά αντικείμενα όπως κοσμήματα, παπούτσια, ρούχα και γυαλιά.

Επιπλέον είναι ενδιαφέρον να αναφερθούμε σε κάποια από τα θετικά και τα αρνητικά στοιχεία της συλλογής πληροφοριών με μέσα όπως κάμερες και μικρόφωνα, και να τα συγκρίνουμε με τους αισθητήρες συλλογής βιοσημάτων που τοποθετούνται στην επιφάνεια του δέρματος. Παρόλο που οι πρώτοι δε περιλαμβάνουν φυσική επαφή και παρέχουν έναν ευνόητο τρόπο επικοινωνίας, μπορεί να θεωρηθεί ότι παραβιάζουν τον 'ιδιωτικό χώρο' του χρήστη. Ο χρήστης ίσως επιθυμεί συναισθηματική επικοινωνία αλλά δεν επιθυμεί να μεταφερθεί η μορφή του ή να καταγραφεί η φωνή του. Επιπλέον, ίσως είναι δύσκολο για έναν απλό χρήστη να απενεργοποιήσει το σύστημα αναγνώρισης φωνής ή το σύστημα της όρασης που είναι εγκατεστημένο σε ένα 'έξυπνο δωμάτιο'. Δημιουργώντας αισθητήρες φυσιολογίας μέσα σε συστήματα που μπορούν να φορεθούν, ή ενσωματώνοντάς τους σε παραδοσιακές συσκευές εισόδου όπως το ποντίκι ή το πληκτρολόγιο, ο χρήστης διατηρεί τον αρχικό έλεγχο. Δηλαδή, διατηρεί την επιλογή να απενεργοποιήσει εύκολα τους αισθητήρες όποτε το επιθυμεί, και μπορεί να είναι σίγουρος ότι αυτά τα σήματα δεν παρέχουν αναγνωριστική πληροφορία όπως κάνουν τα ακουστικά συστήματα αναγνώρισης φωνής ή τα οπτικά συστήματα αναγνώρισης προσώπου.

Φυσικά, είναι σημαντικό να θυμόμαστε ότι ο τύπος των πληροφοριών που αντιλαμβάνεται ένας αισθητήρας βιοσημάτων είναι διαφορετικός από αυτόν που αντιλαμβάνονται και συλλέγουν οι κάμερες ή τα μικρόφωνα.

Σε αυτό το σημείο, θα ήταν χρήσιμο για την πληρότητα της αναφοράς να συνοψίσουμε όσα αναφέρθηκαν παραπάνω και να αναφερθούμε στις δύο μεγάλες κατηγορίες δεδομένων με τη βοήθεια των οποίων μπορεί να γίνει ανίχνευση της συναισθηματικής κατάστασης ενός ατόμου:

2.2.1 Παρεισφρητικά δεδομένα

Τα δεδομένα αυτά αποτελούν βιοσήματα του ανθρώπου και καταδεικνύουν μεταβολές στο συναισθηματικό κόσμο του χρήστη. Παραδείγματα τέτοιων βιοσημάτων που μπορούν να βοηθήσουν τη μελέτη μας είναι η αγωγιμότητα του δέρματος, ο καρδιακός ρυθμός, τα επίπεδα της κορτιζόνης, η αναπνοή, η ηλεκτρική αγωγιμότητα του δέρματος, το ηλεκτροεγκεφαλογράφημα, η διάμετρος της κόρης του ματιού, ο ιδρώτας στις παλάμες και η ένταση των μυών. Τα σήματα αυτά συλλέγονται και καταγράφονται με τη βοήθεια ειδικών αισθητήρων και μπορούν να δώσουν πολύ σημαντικές πληροφορίες ως προς την ένταση και την ποιότητα της εσωτερικής κατάστασης ενός ατόμου. Τα βιοσήματα αυτού του είδους μπορούν εύκολα να διακριτοποιηθούν και τελικά να απεικονισθούν δίνοντας στους ερευνητές δεδομένα για αλλαγές πολύ μικρής διάρκειας που συμβαίνουν κατά τη διάρκεια μίας εργασίας και οι οποίες ίσως να μην είναι μετρήσιμες με άλλους τρόπους.

Παρόλο που υπάρχουν ακόμα διαφωνίες σχετικά με την αξιοπιστία αυτών των σημάτων για συγκεκριμένες συναισθηματικές καταστάσεις, είναι γενικά αποδεκτό ότι τα δεδομένα που πρκύπτουν παρέχουν πληροφορίες τουλάχιστον ως προς το valence και arousal της εσωτερικής κατάστασης του ατόμου και μπορούν να αξιοποιηθούν δρώντας συμπληρωματικά με την όραση υπολογιστών, την ακοή και την επεξεργασία της φυσικής γλώσσας ώστε να γίνουν οι υπολογιστές καλύτεροι γνώστες των συναισθημάτων του χρήστη.

Το αρνητικό στοιχείο αυτής της διαδικασίας, είναι ότι απαιτείται ειδικός εξοπλισμός και τεχνική εκπαίδευση για τη χρήση του. Παρ'όλα αυτά, υπάρχουν κάποια σήματα που μπορούν να καταγραφούν χωρίς να υπάρχει ανάμιξη στην πειραματική διαδικασία, όπως για παράδειγμα, στην περίπτωση που έχουμε καταγραφή του καρδιακού ρυθμού, όπου ο αισθητήρας φοριέται κάτω από τα ρούχα και από τη στιγμή που θα προσκολληθεί στο άτομο δεν απαιτεί κάποια άλλη ιδιαίτερη ενασχόληση.

2.2.2 Μη παρεισφρητικά δεδομένα

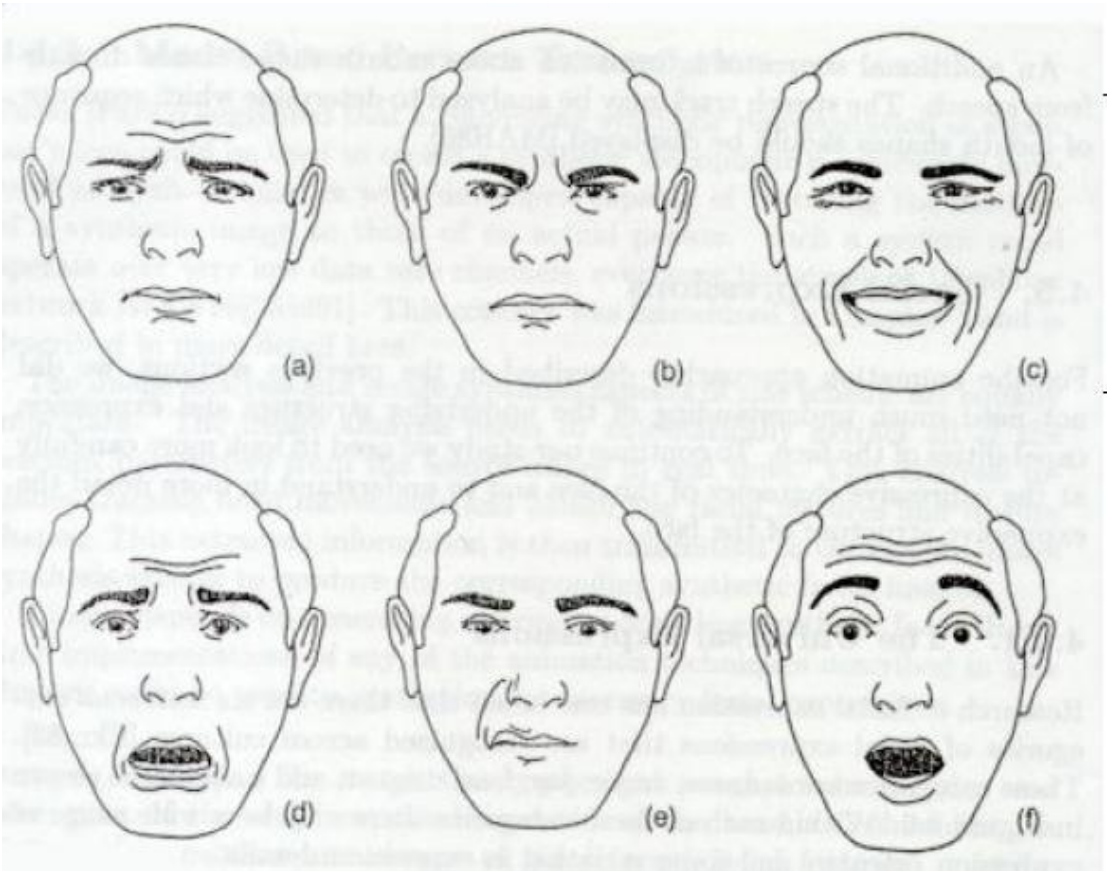
Τα μη παρεισφρητικά δεδομένα που μπορούν να υποδηλώσουν τη συναισθηματική κατάσταση του χρήστη, είναι αυτά που προκύπτουν από τις μεταβολές στη φυσιογνωμία ανάλογα με τα ερεθίσματα του περιβάλλοντος και μπορούν να καταγραφούν με τη βοήθεια απλών μέσων όπως μία κάμερα. Το πλεονέκτημα αυτής της κατηγορίας δεδομένων συνίσταται στην ευκολία καταγραφής τους καθώς απαιτούν στοιχειώδη εξοπλισμό και συνάμα δε δυσχεραίνουν τη ροή του πειράματος. Γι'αυτό το λόγο άλλωστε χρησιμοποιήθηκαν και στη συγκεκριμένη διαδικασία που θα περιγράψουμε στη συνέχεια.

Φυσικά υπάρχουν και μειονεκτήματα βασικότερο εκ των οποίων είναι η υποκειμενικότητα του μελετητή-αναλυτή που υπεισέρχεται στην 'αποκωδικοποίηση' των εκφράσεων και την αντιστοίχισή τους σε συγκεκριμένες συναισθηματικές καταστάσεις.

Επίσης η διαδικασία μπορεί να επηρεαστεί σημαντικά από την πρότερη ψυχολογική κατάσταση του χρήστη που ίσως τον εμποδίσει να ακολουθήσει τη ροή του πειράματος με τον επιθυμητό τρόπο και να μην έχει τις αναμενόμενες αντιδράσεις.

2.3 Ανάλυση δεδομένων – Αναγνώριση συναισθηματικών καταστάσεων

Αναφέρθηκε παραπάνω ότι τα μη παρεισφρητικά δεδομένα, όπως οι εκφράσεις του προσώπου μπορούν να μας οδηγήσουν σε συμπεράσματα αναφορικά με τη συναισθηματική κατάσταση του ατόμου. Για να γίνει περισσότερο κατανοήτη αυτή η διαδικασία θα παραθέσουμε κάποια εποπτικά παραδείγματα που εξηγούν πώς μπορούμε να πετύχουμε κάτι τέτοιο:



Εικόνα 2.1: Βασικές ανθρώπινες εκφράσεις: (α) λύπη, (b) θυμός, (c) χαρά, (d) φόβος, (e) απέχθεια, (f) έκπληξη

Παρατηρώντας την εικόνα 2.1 συμπεραίνουμε ότι λαμβάνοντας υπόψιν κάποιες συγκεκριμένες κινήσεις των τριών εκφραστικών περιοχών του προσώπου που συνίστανται από τα *μάτια*, το *στόμα* και τα *φρύδια*, μπορούμε να οδηγηθούμε σε αρκετά ασφαλή συμπεράσματα αναφορικά με το συναίσθημα που βιώνει τη συγκεκριμένη χρονική στιγμή το άτομο. Σύμφωνα με έρευνες που έχουν γίνει, τα συναισθήματα έχουν κατηγοριοποιηθεί αναλογικά με τις εκφράσεις του προσώπου με τον ακόλουθο τρόπο:

α/α	Έκφραση	Περιγραφή
1	Λύπη	Τα εσωτερικά σημεία των φρυδιών κάμπτονται προς τα πάνω. Τα μάτια είναι ελαφρώς κλειστά. Το στόμα είναι χαλαρωμένο.
2	Θυμός	Τα εσωτερικά σημεία των φρυδιών τραβιούνται προς τα κάτω. Τα μάτια είναι πολύ ανοιχτά. Τα χείλη πιέζονται το ένα στο άλλο ή ανοίγουν και εκθέτουν τα δόντια.
3	Χαρά	Τα φρύδια είναι χαλαρωμένα. Το στόμα είναι ανοιχτό και οι στοματικές γωνίες τραβιούνται πίσω, προς τα αυτιά.
4	Φόβος	Τα φρύδια τραβιούνται μαζί προς τα πάνω. Τα εσωτερικά σημεία των ματιών κάμπτονται προς τα πάνω. Τα μάτια είναι ανήσυχα, άγρυπνα.
5	Απέχθεια	Τα φρύδια και τα βλέφαρα είναι χαλαρωμένα. Το άνω χείλος αυξάνει και σουφρώνει, συχνά ασύμμετρα.
6	Έκπληξη	Τα φρύδια κινούνται προς τα πάνω. Τα άνω βλέφαρα είναι πολύ ανοιχτά, ενώ τα κάτω χαλαρά. Το σαγόνι είναι κατεβασμένο.

Πίνακας 2.1: Χαρακτηριστικά εκφράσεων

Αυτά λοιπόν τα χαρακτηριστικά και άλλα πολυπλοκότερα που δημιουργούνται από σύνθεση δύο ή περισσοτέρων συναισθημάτων, γίνεται προσπάθεια να αναπαρασταθούν και να αποδοθούν σε εικονικούς χαρακτήρες με όσο το δυνατόν μεγαλύτερη ακρίβεια. Παράδειγμα ενός τέτοιου χαρακτήρα αποτελεί η Greta της οποίας παραθέτουμε ενδεικτικά κάποιες εκφράσεις:

Εικόνα 2.2: Απεικόνιση κάποιων βασικών εκφράσεων εικονικού χαρακτήρα: (α) λύπη, (β) θυμός, (γ) φόβος

Βλέπουμε ότι αποδίδονται τα βασικά χαρακτηριστικά του πίνακα 2.1 χωρίς όμως αυτό να αποκλείει τη δυνατότητα εξέλιξης για ακόμα πιο ρεαλιστικά αποτελέσματα.

Για πληρότητα να αναφέρουμε ότι η αντίστοιχη κατηγοριοποίηση υπάρχει και για τα παρεισφρητικά δεδομένα (για παράδειγμα πώς επηρεάζει ο θυμός τον καρδιακό ρυθμό ή την αγωγιμότητα του δέρματος) αλλά δε θα υπάρξει παραπέρα ανάλυση καθώς στα πλαίσια αυτής της εφαρμογής δε θα ασχοληθούμε με δεδομένα αυτού του είδους.

Σε αυτό το σημείο, έχοντας ολοκληρώσει τη θεωρητική ανάλυση περί συναισθηματικής υπολογιστικής, καλούμαστε να αντιμετωπίσουμε το βασικό πρόβλημα αυτής της διαδικασίας που είναι η πρόκληση συναισθημάτων και συναισθηματικών εκφράσεων με αυθόρμητο τρόπο. Η λύση είναι η δημιουργία ενός ηλεκτρονικού παιχνιδιού με συγκεκριμένες και ιδιαίτερες λειτουργίες το οποίο θα αναλύσουμε στη συνέχεια.

3 ΕΙΣΑΓΩΓΗ ΣΤΟ ΠΕΡΙΒΑΛΛΟΝ ΤΟΥ TORQUE

3.1 Εισαγωγή

Για την υλοποίηση της εφαρμογής (ηλεκτρονικό παιχνίδι), που θα μας βοηθήσει στην συλλογή των δεδομένων που αφορούν τη συναισθηματική κατάσταση του χρήστη, χρησιμοποιήθηκε η μηχανή παιχνιδιών Torque και πιο συγκεκριμένα το Torque Game Engine (TGE). Ακολουθεί μία σύντομη περιγραφή της συγκεκριμένης μηχανής όπου αναφέρονται τα βασικά χαρακτηριστικά της.

3.2 Χαρακτηριστικά

Το Torque είναι μία μηχανή παιχνιδιών η οποία παρέχει συμπαγή κώδικα δικτύου, scripting, και δίνει τη δυνατότητα δημιουργίας διαδραστικών εικονικών κόσμων. Το Torque περιέχει μια μηχανή κατασκευής εδάφους (terrain) που δημιουργεί αυτόματα Levels-of-Detail (LODs) έτσι ώστε να αναπαριστά τα ελάχιστα απαραίτητα πολύγωνα οποιαδήποτε στιγμή. Το έδαφος φωτίζεται αυτόματα και τα textures του εδάφους μπορούν να συνδυαστούν μεταξύ τους.
Το Torque υποστηρίζει παιχνίδια μέσω διαδικτύου ή LAN (τοπικό δίκτυο) χρησιμοποιώντας την αρχιτεκτονική πελάτη-εξυπηρετητή (client-server). Τα αντικείμενα του εξυπηρετητή (server) αντιγράφονται στους πελάτες (clients) και ανανεώνονται περιοδικά ή σε κάποια σημεία του παιχνιδιού. Αξίζει να σημειωθεί ότι η εφαρμογή που περιγράφεται στη συγκεκριμένη αναφορά εκμεταλλεύτηκε κατά κύριο λόγο τη δυνατότητα που παρέχει το Torque για ανάπτυξη διαδικτυακών εφαρμογών.

3.3 Πλεονεκτήματα

Μια εκτενής κοινότητα ανάπτυξης ανεξάρτητων παιχνιδιών έχει συσπειρωθεί γύρω από το TGE, εν μέρει και λόγω της χαμηλής τιμής του. Ενώ η ποιότητα του Torque είναι παρόμοια ή ακόμα και χειρότερη από άλλων ελεύθερων, χαμηλού κόστους, ή open-source μηχανών, πολλοί χρήστες θεωρούν ότι το TGE προσφέρει μια πλήρη μηχανή παιχνιδιών, κάτι που άλλες μηχανές του ίδιου κόστους δεν μπορούν να προσφέρουν.

3.3.1 Networking

Το πιο σημαντικό ίσως χαρακτηριστικό του Torque είναι η δυνατότητά του να συνδέεται με άλλα προγράμματα μέσω διαδικτύου. Θεωρητικά έχει εξαιρετικά χαμηλό latency και είναι σε θέση να τρέξει online παιχνίδια χωρίς σημαντική καθυστέρηση. Πέρα όμως από τη δυνατότητα σύνδεσης με άλλους χρήστες του διαδικτύου, το Torque παρέχει εγγενή δομή client-server (πελάτη-εξυπηρετητή) και με αυτό τον τρόπο αφ'ενός επιτρέπει ανάπτυξη εφαρμογών σε επίπεδο τοπικού δικτύου αφ'εταίρου συμβάλλει στην αποσυμφόρηση της δικτυακής κίνησης.

3.3.2 Κοινότητα

Η κοινότητα των χρηστών του TGE συνεισφέρει σημαντικά στη διαμόρφωση του προγράμματος. Τα διάφορα plugins των χρηστών βρίσκονται στην ιστοσελίδα της

GarageGames και μπορεί ο καθένας να τα χρησιμοποιήσει. Έτσι, μεγάλο μέρος της επιτυχίας του Torque βασίζεται σε αυτή τη συμμετοχή των ανεξάρτητων χρηστών.

3.4 Μειονεκτήματα

Πολλοί έχουν εκφράσει την αρνητική κριτική τους για το TGE. Μεταξύ άλλων, αναφέρονται στην κακή μορφή των τρισδιάστατων αντικειμένων, στο κακό documentation (εγχειρίδιο βοήθειας), στην κατώτερη ηχητική υποστήριξη σε σύγκριση με το tribes 2, στην έλλειψη map editors, στα ξεπερασμένα γραφικά, και στον ανοργάνωτο κώδικα.

3.4.1 Documentation

Όσοι κριτικάρουν το Torque συχνά αναφέρουν το documentation ως μια από τις μεγάλες αδυναμίες του. Ενώ το μέγεθος του documentation είναι ικανοποιητικό, η ποιότητά του είναι κακή και οι πληροφορίες που περιέχει λίγες, ειδικά αν συνυπολογίσουμε το γεγονός ότι το TGE έχει αρκετές ιδιαιτερότητες που δεν μπορούν να κατανοηθούν με απλή παρατήρηση του κώδικα. Η GarageGames έχει καταβάλει προσπάθειες να μετριαστεί η κριτική με τη δημιουργία του Torque Development Network που είναι ένα κλειστό wiki για όσους έχουν αγοράσει τη μηχανή.

3.4.2 Ήχος

Η υποστήριξη που έχει το Torque στον τομέα του ήχου είναι σημαντικά ασθενέστερη από την αντίστοιχη στο tribes 2, απ' όπου ξεκίνησε. Για να διατηρήσει η GarageGames τη χαμηλή τιμή για το TGE, έπρεπε να χρησιμοποιήσει την μόνη cross-platform ηχητική βιβλιοθήκη: το OpenAL. Όμως η ποιότητα του OpenAL και η υποστήριξή του είναι σημαντικά χαμηλότερες από άλλες ηχητικές βιβλιοθήκες.

3.5 Torque Game Engine (TGE)

Όπως στην περίπτωση των περισσότερων μηχανών δημιουργίας παιχνιδιών, το Torque περιλαμβάνει τα παρακάτω τμήματα: τμήμα τρισδιάστατων γραφικών, τμήμα ήχου, τμήμα τεχνητής νοημοσύνης, τμήμα ανίχνευσης σύγκρουσης, τμήμα εισόδου-εξόδου, τμήμα βάσης δεδομένων, τμήμα δικτύου και τμήμα γραφικού περιβάλλοντος.
Το Torque υποστηρίζει κώδικα σε μορφή script. Το **TorqueScript** (όπως αποκαλείται), ακολουθεί το αντικειμενοστρεφές μοντέλο και υποστηρίζει τον ορισμό κλάσεων και DataBlocks. Κάθε αντικείμενο που κατασκευάζεται με χρήση της δεσμευμένης λέξης new ανήκει σε μία κλάση. Τα χαρακτηριστικά του αντικειμένου καθορίζονται από τις τιμές των πεδίων της κλάσης και η συμπεριφορά του από της μεθόδους που υλοποιεί. Ορισμένα αντικείμενα χρησιμοποιούν DataBlocks προκειμένου να αποκτήσουν επιπρόσθετες ιδιότητες ή να χρησιμοποιήσουν συναρτήσεις κοινές με άλλα αντικείμενα. Το TorqueScript υποστηρίζει επιπλέον την κληρονομικότητα και τον πολυμορφισμό. Το συντακτικό της γλώσσας μοιάζει με αυτό της C++ με κύρια διαφορά το ότι στο TorqueScript δεν απαιτούνται δηλώσεις τύπων των μεταβλητών που χρησιμοποιούνται. Η script language κάνει διάκριση μεταξύ τοπικών και παγκόσμιων μεταβλητών (οι

τοπικές μεταβλητές έχουν το πρόθεμα % ενώ οι παγκόσμιες το πρόθεμα $) και χρησιμοποιεί τις βασικότερες δομές ελέγχου ροής του προγράμματος και τους αριθμητικούς και λογικούς τελεστές, που χρησιμοποιεί και η C++.
Επίσης υποστηρίζει δυναμικό φόρτωμα πακέτων του κώδικα όταν αυτό απαιτείται από την τρέχουσα εφαρμογή. Ο κώδικας του Torque περιλαμβάνει τον ορισμό περίπου 420 έτοιμων συναρτήσεων, 680 μεθόδων και 200 callbacks. Εκτός από τις προκαθορισμένες συναρτήσεις και μεθόδους του Torque δίνεται η δυνατότητα στον χρήστη να ορίσει και να χρησιμοποιήσει τις δικές του συναρτήσεις και μεθόδους. Μέσω κώδικα γραμμένου σε TorqueScript μπορούν να ελεγχθούν όλες οι λειτουργίες των εφαρμογών που εκτελούνται στο Torque Game Engine, όπως η δημιουργία και καταστροφή αντικειμένων, ανίχνευση εισόδου και ανταπόκριση σε αυτή, ανίχνευση σύγκρουσης μεταξύ αντικειμένων, δημιουργία συνδέσεων σε εφαρμογές που ακολουθούν την αρχιτεκτονική πελάτη-εξυπηρετητή, κατασκευή γραφικού περιβάλλοντος, καθορισμός θέσης, περιστροφής και κλίμακας των αντικειμένων που εμφανίζονται στο περιβάλλον της εφαρμογής, καθορισμός animation και ταχύτητας κίνησης χαρακτήρων, χειρισμός ήχου, οργάνωση χρονοδιαγράμματος και ορισμός ακολουθιών διαδοχικών ενεργειών.
Εκτός από το TorqueScript, το Torque Game Engine έχει κάποιες στοιχειώδεις εφαρμογές για την κατασκευή του ψηφιακού κόσμου και του γραφικού περιβάλλοντος. Αυτές οι εφαρμογές είναι ο **world editor** και ο **GUI editor** αντίστοιχα.

Ο **world editor** χρησιμοποιείται για τον σχεδιασμό του περιβάλλοντος της εφαρμογής και περιλαμβάνει τον **terrain editor** για την διαμόρφωση του ‘εδάφους’ της εφαρμογής. Η επιλογή του **world inspector** εμφανίζει μια δενδρική δομή στο δεξί μέρος της οθόνης, όπου εμφανίζονται όλα τα αντικείμενα που αποτελούν το περιβάλλον του εικονικού κόσμου συνοδευόμενα από το ID τους. Με επιλογή οποιουδήποτε αντικειμένου της δενδρικής δομής εμφανίζονται πληροφορίες που αφορούν την θέση του, την περιστροφή του, το μέγεθός του και μία σειρά άλλων χαρακτηριστικών ιδιοτήτων, ανάλογα με το είδος του αντικειμένου. Τα αντικείμενα του περιβάλλοντος μπορούν να ομαδοποιηθούν με χρήση αντικειμένων SimGroup που λειτουργούν σαν φάκελοι και συμβάλλουν στην καλύτερη οργάνωση της δενδρικής δομής του εικονικού κόσμου όπως παρουσιάζεται και στο παρακάτω παράδειγμα:

Εικόνα 3.1: Οργάνωση των αντικειμένων του παιχνιδιού σε SimGroups και κατηγορίες αντικειμένων

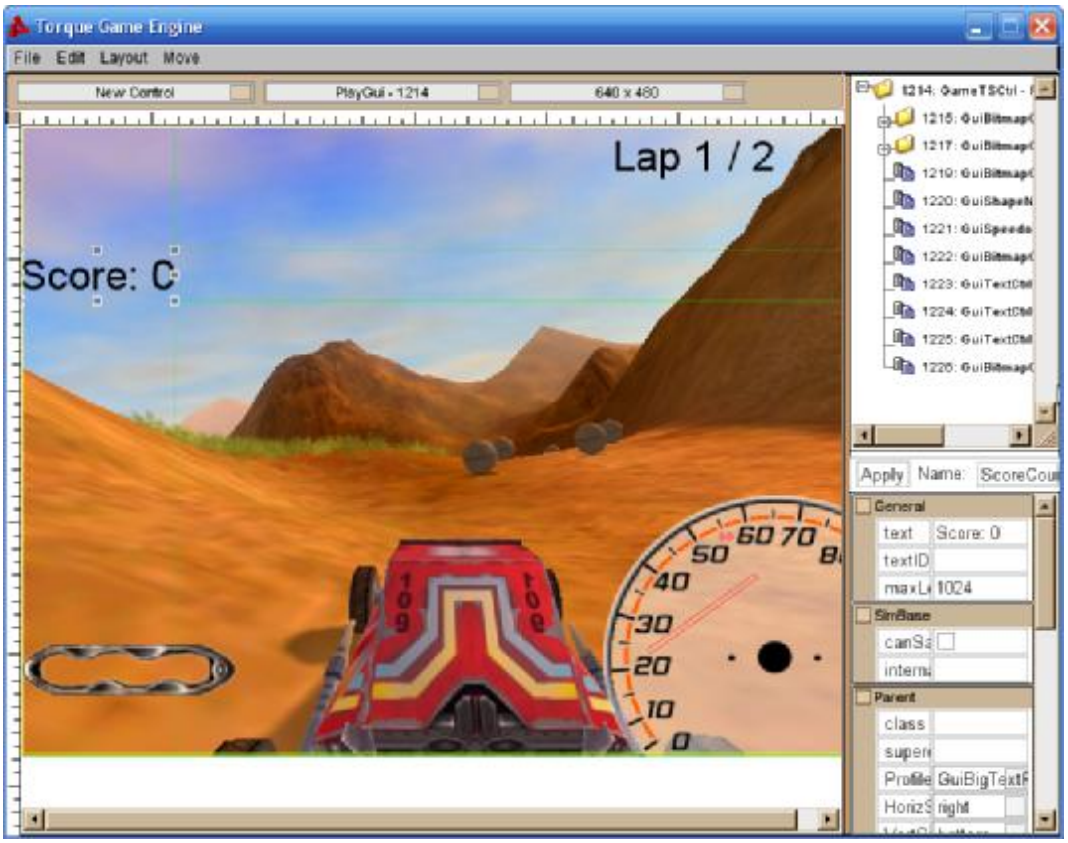
Τα αντικείμενα που απαρτίζουν το περιβάλλον του εικονικού κόσμου κατασκευάζονται μέσω του **world creator**. Ο world creator χωρίζει τα αντικείμενα αυτά σε 4 κατηγορίες όπως φαίνεται και στην εικόνα 3.1: αντικείμενα τύπου Interiors, Shapes, Static Shapes και Mission Objects. Μετά την κατασκευή ενός αντικειμένου δίνεται η δυνατότητα στον χρήστη να ορίσει την θέση που θα καταλαμβάνει στον εικονικό κόσμο, το όνομά του και τα υπόλοιπα χαρακτηριστικά του. Ο world editor παρέχει επιπλέον επιλογές που έχουν να κάνουν με τη θέση, τη λήψη και την ταχύτητα κίνησης της κάμερας και περιλαμβάνονται στο μενού camera. Μετά την τοποθέτηση ή την αλλαγή της θέσης οποιουδήποτε αντικειμένου είναι απαραίτητη η επιλογή του ‘relight scene’ που υπολογίζει τις σκιές που δημιουργεί το νέο αντικείμενο στο περιβάλλον του. Μία εποπτική εικόνα του world editor της εφαρμογής που δημιουργήσαμε φαίνεται στη συνέχεια:

Εικόνα 3.2: Ο World editor της εφαρμογής

Ο **terrain editor**, όπως προαναφέρθηκε, χρησιμοποιείται για την διαμόρφωση του ‘εδάφους’ της εφαρμογής. Κατά την επιλογή του εμφανίζεται το εργαλείο ***brush*** που επιτρέπει την διαμόρφωση της τοπολογίας της εφαρμογής με δημιουργία υψωμάτων στο έδαφος. Το μέγεθος του εργαλείου brush, η ισχύς του και το σχήμα του ρυθμίζονται μέσω των επιλογών του μενού brush. Το εργαλείο **terrain terraform editor** χρησιμοποιείται για την αλγοριθμική επεξεργασία του terrain. Συγκεκριμένα ο terrain terraform editor παρέχει μια σειρά τελεστών και φίλτρων που αν εφαρμοστούν στο αρχικό terrain τροποποιούν την μορφή του. Για την επιλογή textures του terrain υπάρχουν τα εργαλεία **terrain texture editor** και **terrain texture painter**. Συγκεκριμένα μέσω του terrain texture editor καθορίζεται ο τρόπος με τον οποίο θα εφαρμοστεί το texture (υφή) στο terrain, προσδίδοντας στην εμφάνισή του περισσότερες λεπτομέρειες. Ο terrain texture painter επιτρέπει την επιλογή μέχρι και 6 διαφορετικών textures τα οποία μπορούν να εφαρμοστούν σε διαφορετικά σημεία του terrain.

Ο **GUI editor** είναι η εφαρμογή που επιτρέπει τον σχεδιασμό του γραφικού περιβάλλοντος το οποίο χρησιμοποιείται για την επικοινωνία του χρήστη με την εφαρμογή. Μέσω του GUI editor δίνεται η δυνατότητα κατασκευής αντικειμένων που αποτελούν συστατικά στοιχεία του γραφικού περιβάλλοντος. Επίσης είναι δυνατός ο προσδιορισμός της θέσης τους, του μεγέθους τους και των υπολοίπων χαρακτηριστικών τους όπως το αν θα είναι ορατά ή όχι καθώς και οι ενέργειες που θα εκτελούνται κατά την επιλογή τους, αν πρόκειται για αλληλεπιδραστικά αντικείμενα. Επίσης καθορίζεται

το μέγεθος του παραθύρου της εφαρμογής και ο τρόπος με τον οποίο θα προσαρμοστεί το μέγεθος και η θέση των στοιχείων του GUI όταν μεταβάλλεται το μέγεθος του παραθύρου που τα περιέχει. Ο GUI editor παρέχει μια δενδρική δομή στην οποία εμφανίζονται όλα τα στοιχεία του γραφικού περιβάλλοντος. Με επιλογή κάποιου αντικειμένου της δενδρικής δομής εμφανίζονται οι ιδιότητες του αντικειμένου αυτού.



Εικόνα 3.3: Ο GUI editor της εφαρμογής

Όλα αυτά τα χαρακτηριστικά του Torque αξιοποιήθηκαν για την υλοποίηση της εφαρμογής της οποίας η δομή θα αναλυθεί στη συνέχεια.

4 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

4.1 Εισαγωγή

Στο κεφάλαιο που ακολουθεί θα δοθεί αναλυτικά η δομή της εφαρμογής που αναπτύχθηκε για τις ανάγκες του πειράματός μας. Τη βάση της εφαρμογής αποτέλεσε η έτοιμη πλατφόρμα του παιχνιδιού starter.racing του TGE, το οποίο εμπλουτίστηκε με αρκετά στοιχεία ώστε να γίνει ένα ολοκληρωμένο ηλεκτρονικό παιχνίδι πάνω στο οποίο εφαρμόστηκαν οι διάφοροι τρόποι παρεμβολής που θα αναφερθούν και θα αναλυθούν σε επόμενο κεφάλαιο της αναφοράς. Το νέο αυτό ηλεκτρονικό παιχνίδι ονομάστηκε **'Frustracing'**. Πριν προχωρήσουμε όμως στη δομή της εφαρμογής θα αναφερθούμε σε κάποια απαραίτητα γενικά χαρακτηριστικά του Torquescript.

4.2 Οι έννοιες του αντικειμένου και του DataBlock

Τα αντικείμενα (objects) είναι στιγμιότυπα κλάσεων αντικειμένων (object classes), οι οποίες αποτελούν μια συλλογή ιδιοτήτων και μεθόδων που καθορίζουν ένα συγκεκριμένο σύνολο συμπεριφορών και χαρακτηριστικών. Μετά από τη δημιουργία του, ένα αντικείμενο παίρνει ένα μοναδικό αριθμό που αποκαλείται handle. Όταν δύο handles έχουν την ίδια τιμή, αναφέρονται στο ίδιο αντικείμενο. Όταν ένα αντικείμενο υπάρχει σε μια εφαρμογή πολλαπλών χρηστών με έναν κεντρικό εξυπηρετητή και πολλούς πελάτες, ο εξυπηρετητής και κάθε πελάτης διαθέτει το handle για την αποθήκευση του αντικειμένου στη μνήμη. Να σημειωθεί ότι τα dataBlocks (ένα ειδικό είδος αντικειμένου) αντιμετωπίζονται διαφορετικά.

4.2.1 Δημιουργία ενός αντικειμένου

Όταν δημιουργούμε το στιγμιότυπο ενός αντικειμένου, μπορούν να αρχικοποιηθούν πολλές μεταβλητές του αντικειμένου μέσα στον κώδικα new:

Παράδειγμα:

```
new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-166.289 974.153 193.296";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
   };
```

Οι βασικές παράμετροι του αντικειμένου είναι οι Position (θέση), Rotation (γωνία), Scale (μέγεθος) και dataBlock. Αυτές οι παράμετροι είναι απαραίτητες για να εμφανιστεί ένα αντικείμενο StaticShape στην εφαρμογή. Εκτός από αυτές όμως, υπάρχουν και κάποιες επιπλέον παράμετροι οι οποίες χρησιμοποιούνται από την εφαρμογή για να καθοριστούν ειδικές ιδιότητες που χαρακτηρίζουν τη συμπεριφορά του αντικειμένου στο γραφικό περιβάλλον.

4.2.2 Χρήση αντικειμένου

Για να χρησιμοποιήσουμε ένα αντικείμενο, μπορούμε να κάνουμε χρήση του handle του για να αποκτήσουμε πρόσβαση στις παραμέτρους του και τις συναρτήσεις του ή ακόμα μπορούμε να χρησιμοποιήσουμε και το όνομα του αντικειμένου, αρκεί το όνομα αυτό να είναι μοναδικό.

4.2.3 Συναρτήσεις αντικειμένων

Ο τρόπος που ορίζεται μια συνάρτηση για ένα συγκεκριμένο αντικείμενο είναι η εξής:

function narki::onCollision(%this, arg1, arg1, arg3){}

Η συνάρτηση αυτή ορίζει τη διαδικασία της σύγκρουσης (παράμετροι που επηρεάζονται, γεγονότα που συμβαίνουν) για το αντικείμενο ‘narki’.
Ως επί το πλείστον όμως, οι συναρτήσεις ορίζονται για κλάσεις και όχι για αντικείμενα, και επειδή οι κλάσεις είναι σύνολα από DataBlocks, όπως θα αναλύσουμε παρακάτω, για να κληθεί μια τέτοια συνάρτηση πρέπει να γραφεί το εξής:

%this.getDataBlock().damage(arg1, arg1, arg3){}

Όταν καλείται μια συνάρτηση, το πρώτο όρισμα είναι το handle του αντικειμένου που περιέχει η συνάρτηση. Γι’ αυτό και στον παραπάνω ορισμό της συνάρτησης υπάρχει ένα παραπάνω όρισμα, το %this. Το όρισμα %this δεν χρησιμοποιείται όταν καλούμε τη συνάρτηση, αλλά προστίθεται από την μηχανή αυτόματα.

4.2.4 DataBlocks

Το DataBlocks είναι ένας ειδικός τύπος αντικειμένου που περιέχει ένα σύνολο χαρακτηριστικών που χρησιμοποιούνται για να περιγράψουν τις ιδιότητες ενός άλλου αντικειμένου. Υπάρχουν στον εξυπηρετητή και φορτώνονται από αυτόν σε όλους τους πελάτες κατά την έναρξη της αποστολής.
Τα DataBlocks ανήκουν σε κάποιες κατηγορίες (PlayerData, StaticShapeData, ItemData κτλ.), ανάλογα και με το είδος των αντικειμένων που περιγράφουν (Player, StaticShape, Item κτλ). Κάθε κατηγορία έχει διαφορετικές παραμέτρους που μπορούν να οριστούν. Για παράδειγμα, το PlayerData έχει πολλές παραμέτρους για την ταχύτητα του avatar, τη μέγιστη γωνία που μπορεί να ανέβει κτλ. Αντίθετα, στο StaticShapeData αρκεί να οριστεί η κλάση του DataBlock και το αρχείο του μοντέλου που το απεικονίζει. Όταν ένα

αντικείμενο ‘αντιστοιχίζεται’ σε ένα DataBlock, κληρονομεί όλες τις παραμέτρους του και τις συναρτήσεις του (εκτός αν ορίσει δικές του παραμέτρους και συναρτήσεις, με τα ίδια ονόματα με τις αντίστοιχες του DataBlock).
Παρόλο που τα αντικείμενα μπορεί να δημιουργούνται και να διαγράφονται συνεχώς μέσα σε μια εφαρμογή, τα DataBlocks δημιουργούνται όταν φορτώνεται η αποστολή και δεν διαγράφονται. Ο τρόπος που συντάσσεται το DataBlock είναι ο εξής:

```
datablock StaticShapeData (healthkit)
{
        shapeFile = "~/data/shapes/buggy/firstaid.dts";
        category = "Items";
        mass=1;
        friction=1;
        emap = true;
};
```

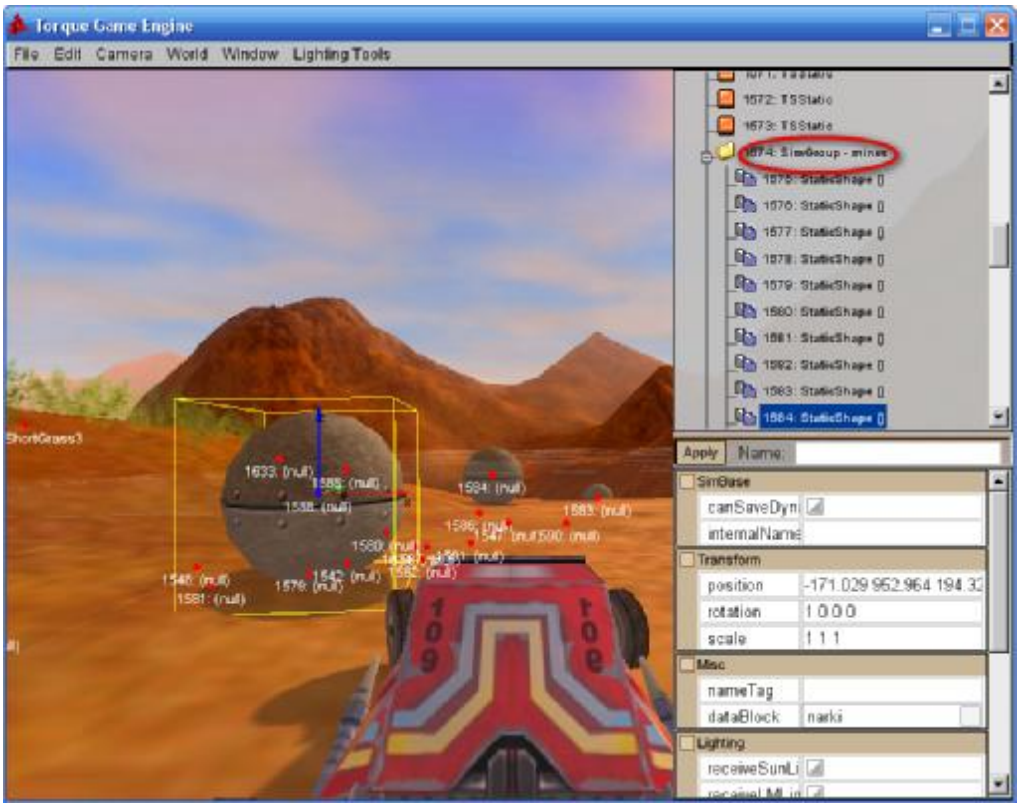
Παρατηρούμε ότι το DataBlock με όνομα healthkit είναι StaticShapeData και άρα τα αντικείμενα που περιγράφει πρέπει να είναι StaticShape. Αναφέρεται η ακριβής θέση του αρχείου μέσα στην εφαρμογή μέσω της παραμέτρου shapeFile. Επίσης υπάρχει η επιπλέον παράμετρος category, που χρησιμοποιείται για να ομαδοποιούνται παρόμοια DataBlocks στην εφαρμογή. Όπως φαίνεται στο παρκάτω στιγμιότυπο έχουν οριστεί πράγματι τα αντικείμενα ‘narki’ και ‘healthkit’ να ανήκουν στην κατηγορία Items.

Εικόνα 4.1: Κατηγοριοποίηση των αντικειμένων της εφαρμογής

4.2.5 Αντικείμενα που συμβάλλουν στην οργάνωση της εφαρμογής

Μέχρι τώρα αναφερθήκαμε σε αντικείμενα που τοποθετούνται στο γραφικό περιβάλλον της εφαρμογής μας και ο χρήστης αλληλεπιδρά με αυτά. Υπάρχουν όμως και τα αντικείμενα τα οποία δεν τοποθετούνται στο περιβάλλον της εφαρμογής και έχουν σκοπό την οργάνωση των υπόλοιπων αντικειμένων του περιβάλλοντος και την αποθήκευση πληροφοριών σχετικά με το είδος της εφαρμογής. Τέτοια αντικείμενα είναι αυτά της κλάσης SimGroup στα οποία έχουμε αναφερθεί και σε προηγούμενο κεφάλαιο. Τα στιγμιότυπα της κλάσης SimGroup λειτουργούν σαν φάκελοι που βοηθούν στην ομαδοποίηση παρεμφερών αντικειμένων και κατ' επέκταση στην αναγνωσιμότητα του αρχείου data\missions\racing.mis που περιέχει τα συστατικά στοιχεία του περιβάλλοντος της εφαρμογής. Επίσης διευκολύνουν τον προγραμματισμό της εφαρμογής, αφού με την βοήθειά τους μπορούν να ομαδοποιηθούν τα αντικείμενα του εικονικού κόσμου και στη συνέχεια τα περιεχόμενα της κάθε ομάδας αντιμετωπίζονται με παρόμοιο τρόπο. Για παράδειγμα, στην περίπτωση του SimGroup 'mines' που παρουσιάζεται στη συνέχεια, όλα τα αντικείμενα αυτής της ομάδας έχουν κοινές ιδιότητες ως προς τη σύγκρουσή τους με το όχημα του παίχτη και τις συνέπειες αυτής της σύγκρουσης (πχ διαγραφή τους από το γραφικό περιβάλλον).
Παράδειγμα SimGroup που χρησιμοποιήθηκε στην εφαρμογή:

```
new SimGroup(mines) {
    canSaveDynamicFields = "1";
```



Εικόνα 4.2: Οργάνωση αντικειμένων σε SimGroups

4.3 Το αρχείο της αποστολής

Η πλειονότητα των αντικειμένων της εφαρμογής βρίσκονται στο αρχείο αποστολής (με κατάληξη .mis) το οποίο βρίσκεται στο φάκελο data/missions. Τα αντικείμενα τηςαποστολής που καταχωρήθηκαν στο αρχείο αποστολής είναι είτε TSStatic, είτε StaticShape, καθώς και μερικά ειδικά αντικείμενα.

4.3.1 TSStatic

Αντικείμενα που ανήκουν στην κατηγορία TSStatic είναι στατικά και δεν είναι αλληλεπιδραστικά. Σε αντίθεση με τα StaticShape αντικείμενα που θα αναλύσουμε παρακάτω, τα TSStatic δεν έχουν DataBlocks ούτε συναρτήσεις. Παραδείγματα TSStatic αντικειμένων στην εφαρμογή είναι τα δέντρα και οι θάμνοι που απλά υπάρχουν και δεν αλληλεπιδρούν καθόλου με το χρήστη.
Παράδειγμα:

```
new TSStatic() {
    canSaveDynamicFields = "1";
    position = "-211.945 935.638 209.513";
    rotation = "1 0 0 15.4698";
    scale = "1 1 1";
    shapeName = "~/data/shapes/trees/treea_1.dts";
    receiveSunLight = "1";
    receiveLMLighting = "1";
    useAdaptiveSelfIllumination = "0";
    useCustomAmbientLighting = "0";
    customAmbientSelfIllumination = "0";
    customAmbientLighting = "0.8 0.7 0.6 1";
  };
```

4.3.2 StaticShape

Τα αντικείμενα της κατηγορίας StaticShape ανήκουν πάντα σε ένα DataBlock στο οποίο υπάρχουν οι παράμετροι που τα προσδιορίζουν. Όλα τα StaticShapes είναι αλληλεπιδραστικά, που σημαίνει ότι ο χρήστης μπορεί να τα χρησιμοποιήσει. Τέλος, έχουν τις απαραίτητες παραμέτρους (θέση, γωνία, μέγεθος, DataBlock) αλλά συχνά περιλαμβάνουν και επιπλέον παραμέτρους που χρησιμοποιεί ο κώδικας για ειδικές χρήσεις.
Παράδειγμα:

```
new StaticShape() {
       canSaveDynamicFields = "1";
       position = "-227.826 490.199 150.383";
       rotation = "1 0 0 0";
       scale = "1 1 1";
```

```
        dataBlock = "narki";
        receiveSunLight = "1";
        receiveLMLighting = "1";
        useAdaptiveSelfIllumination = "0";
        useCustomAmbientLighting = "0";
        customAmbientSelfIllumination = "0";
        customAmbientLighting = "0 0 0 1";
     };
```

Βλέπουμε ότι το συγκεκριμένο StaticShape περιγράφεται από το datablock ‘narki’ το οποίο ορίζεται στο αρχείο server/scripts/narki.cs ως εξής:

```
datablock StaticShapeData (narki)
{
          shapeFile = "~/data/shapes/buggy/simpleshape.dts";
          category = "Items";
          mass=1;
          friction=1;
          emap = true;
          damageAmount =20;
          explosion = narkiExplosion;
};
```

Οι υπόλοιπες παράμετροι αφορούν τη θέση, την περιστροφή, την κλίμακα καθώς και ρυθμίσεις σχετικά με το φωτισμό του αντικειμένου μέσα στο χώρο της εφαρμογής.

4.3.3 Ειδικά αντικείμενα

Τα ειδικά αντικείμενα περιγράφουν όλη την αποστολή και δεν απεικονίζουν κάτι συγκεκριμένο.

```
new ScriptObject(MissionInfo) {
        desc0 = "This is a simple racing example mission.";
        name = "Racing Example";
        type = "racing";
   };

new MissionArea(MissionArea) {
      canSaveDynamicFields = "1";
      Area = "-1024 -1024 2048 2048";
      flightCeiling = "300";
      flightCeilingRange = "20";
      locked = "true";
   };

new Sun() {
```

```
   canSaveDynamicFields = "1";
   azimuth = "50";
   elevation = "30";
   color = "0.6 0.5 0.45 1";
   ambient = "0.4 0.3 0.3 1";
   CastsShadows = "1";
   scale = "1 1 1";
   direction = "0.57735 0.57735 -0.57735";
   position = "0 0 0";
   rotation = "1 0 0 0";
};
```

- Το **ScriptObject** απλά περιέχει μια περιγραφή της αποστολής και το όνομά της.
- Το **MissionArea** εκφράζει τα όρια της αποστολής τόσο στην οριζόντια επιφάνεια (area) όσο και στο μέγιστο ύψος (flightCeiling). Τα ουσιαστικά όρια της αποστολής είναι πολύ πιο περιορισμένα, αλλά ο ορισμός του MissionArea είναι απαραίτητος και γι' αυτό παρατίθεται.
- Το **Sun** είναι κατά μια έννοια ένα αντικείμενο, αλλά δεν αντιπροσωπεύει τόσο τον ήλιο όσο μια γενική πληροφορία για το διάχυτο φως της εφαρμογής. Η κατεύθυνση των ακτίνων φωτός καθορίζει ποιές επιφάνειες θα είναι πιο σκούρες και ποιές πιο φωτισμένες. Επίσης υπάρχει πληροφορία για το χρώμα του φωτός αλλά και το χρώμα της 'σκιάς' (ambient), πόσο πιο σκούρα θα είναι δηλαδή μια επιφάνεια που δεν φωτίζεται.

Το ολοκληρωμένο άρχειο της αποστολής racing.mis θα παρουσιαστεί στο παράρτημα της αναφοράς.

4.3.4 Αντικείμενα ελέγχου

Πέραν των αντικειμένων που διαμορφώνουν το ορατό αποτέλεσμα του εικονικού κόσμου, εξίσου σημαντικά είναι και τα αντικείμενα ελέγχου τα οποία συνήθως δεν είναι ορατά από τον χρήστη κατά την διάρκεια εκτέλεσης της εφαρμογής. Η ύπαρξη των αντικειμένων αυτών έχει ως σκοπό την αποθήκευση πληροφοριών σε συγκεκριμένα σημεία του περιβάλλοντος για τον έλεγχο των δραστηριοτήτων και τον καθορισμό της ροής εκτέλεσης της εφαρμογής.
Τέτοια αντικείμενα είναι τα triggers, τα αντικείμενα της κλάσης SpawnSphere, τα paths και οι markers.

4.3.4.1 Area triggers

Τα χωρικά triggers που χρησιμοποιούνται στην εφαρμογή, οριοθετούν μία περιοχή του εικονικού κόσμου στην οποία προσδίδονται στην συνέχεια συγκεκριμένες ιδιότητες και καθορισμένη συμπεριφορά κατά την αλληλεπίδρασή της με τους χαρακτήρες της εφαρμογής. Οι μέθοδοι onEnterTrigger, onLeaveTrigger και onTickTrigger καθορίζουν τις ενέργειες που εκτελούνται κάθε φορά που κάποιος χαρακτήρας ή αντικείμενο εισέρχεται στον χώρο του trigger, εξέρχεται ή παραμένει σε αυτόν, αντίστοιχα.

Συγκεκριμένα για κάθε trigger κατασκευάζεται ένα νέο αντικείμενο που αποτελεί στιγμιότυπο της κλάσης Trigger και περιλαμβάνει πληροφορίες σχετικά με το σχήμα του, την θέση του αντικειμένου στον εικονικό κόσμο, την κλίση του ως προς κάθε άξονα και το σχετικό μέγεθός του, μέσω των τιμών των παραμέτρων polyhedron, position, rotation και scale, αντίστοιχα. Επίσης, στο αντικείμενο αυτό αντιστοιχίζεται το datablock που θα καθορίσει την συμπεριφορά του μέσω της τιμής της μεταβλητής dataBlock.
Παράδειγμα κώδικα των triggers που χρησιμοποιούνται στην εφαρμογή μας παρουσιάζεται παρακάτω:

```
new Trigger(checkpoint4) {
     canSaveDynamicFields = "1";
     position = "-197.895 872.003 186.979";
     rotation = "1 0 0 0";
     scale = "60 5 20";
     dataBlock = "CheckPointTrigger";
     polyhedron = "0.0000000 0.0000000 0.0000000 1.0000000 0.0000000 0.0000000
0.0000000 -1.0000000 0.0000000 0.0000000 0.0000000 1.0000000";
        isLast = "1";
        checkpoint = "4";
   };
   new Trigger(checkpoint1) {
     canSaveDynamicFields = "1";
     position = "-286.543 1041.54 156.512";
     rotation = "0 0 1 26.356";
     scale = "60 5 20";
     dataBlock = "CheckPointTrigger";
     polyhedron = "0.0000000 0.0000000 0.0000000 1.0000000 0.0000000 0.0000000
0.0000000 -1.0000000 0.0000000 0.0000000 0.0000000 1.0000000";
        checkpoint = "1";
   };
```

Τα Triggers χρησιμοποιούνται στην παρούσα εφαρμογή για να υποδείξουν στο χρήστη ότι κινείται ανάποδα στο χώρο της αποστολής. Εάν δηλαδή το όχημα του παίχτη έρθει σε επαφή με την περιοχή όπου είναι τοποθετημένα τα triggers με την αντίθετη κατεύθυνση από αυτή που θα έπρεπε, θα εμφανιστεί μήνυμα που θα προειδοποιεί για τη λάθος πορεία:

Εικόνα 4.3: Λειτουργία αντικειμένου Trigger

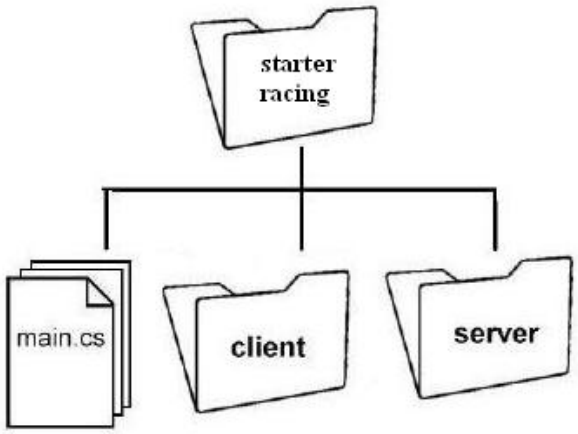
4.3.4.2 SpawnSpheres

Τα αντικείμενα της κλάσης SpawnSphere καθορίζουν σημεία του εικονικού κόσμου που μπορούν να χρησιμεύσουν ως σημεία εμφάνισης των χαρακτήρων της εφαρμογής.

```
new SpawnSphere() {
    canSaveDynamicFields = "1";
    position = "-169.119 928.525 194.926";
    rotation = "0 0 -1 2.47371";
    scale = "1 1 1";
    dataBlock = "SpawnSphereMarker";
    Radius = "1";
    sphereWeight = "100";
    indoorWeight = "100";
    outdoorWeight = "100";
      locked = "False";
      lockCount = "0";
      homingCount = "0";
  };
```

4.4 Η δομή της εφαρμογής

Πριν προχωρήσουμε στη λεπτομερή ανάλυση της λειτουργίας της εφαρμογής που πραγματεύεται η συγκεκριμένη αναφορά, θα ήταν χρήσιμο να παρουσιάσουμε τη δομή της, πώς δηλαδή οργανώνονται όλα τα αρχεία που περιέχονται σε αυτήν και ποιά είναι η λειτουργία τους.
Ανοίγοντας το φάκελο του παιχνιδιού starter.racing, αντικρίζουμε την παρακάτω δομή:



Εικόνα 4.4: Βασική δομή φακέλου starter.racing

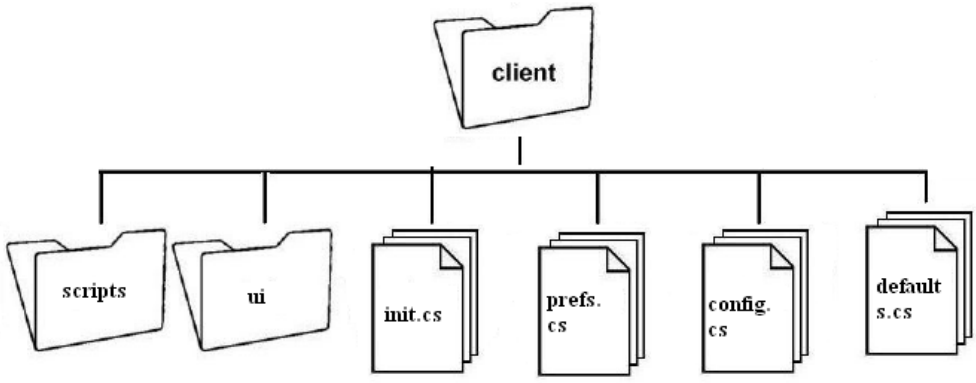
Τα αρχεία είναι οργανωμένα στους φακέλους client και server ενώ υπάρχει και το αρχείο main.cs το οποίο είναι το πρώτο που τρέχει όταν εκτελείται η εφαρμογή. Φυσικά για κάθε αρχείο με κατάληξη .cs που δημιουργείται, υπάρχει και το αντίστοιχο μεταγλωττισμένο αρχείο .cs.dso. Στο αρχείο main.cs εκτελείται αρχικά η συνάρτηση onStart() και 'καλούνται' τα αρχεία ./client/init.cs, ./client/scripts/game.cs, ./server/init.cs και ./data/init.cs στα οποία ορίζονται οι απαραίτητες αρχικοποιήσεις για να ξεκινήσει το παιχνίδι. Το τμήμα αυτό του κώδικα του αρχείου main.cs φαίνεται στη συνέχεια:

```
function onStart()
{
  Parent::onStart();
  echo("\n--------- Initializing MOD: Starter Racing ---------");

  // Φόρτωμα των αρχείων από τα οποία ξεκινάνε τα πάντα
  exec("./client/init.cs");
  exec("./client/scripts/game.cs");
  exec("./server/init.cs");
  exec("./data/init.cs");

  // Ξεκινάει ο εξυπηρετητής
  initServer();
```

```
  // Ξεκινάει είτε λειτουργία πελάτη είτε λειτουργία αφοσιωμένου εξυπηρετητή
  if ($Server::Dedicated)
    initDedicated();
  else
    initClient();
}
```

Στη συνέχεια η κλήση των αρχείων είναι 'αλυσιδωτή' καθώς καθένα από τα αρχεία που κλήθηκαν από το αρχικό, καλεί με τη σειρά του άλλα αρχεία προς εκτέλεση μέχρι να κληθεί και το τελευταίο αρχείο που απαιτεί η εφαρμογή.
Όπως έχει ήδη αναφερθεί, ένα από τα σημαντικά πλεονεκτήματα της μηχανής του Torque, είναι η οργάνωση των αρχείων σε φακέλους πελάτη (*client*), εξυπηρετητή (*server*) και δεδομένων (*data*). Ο φάκελος του server περιέχει όλα τα αρχεία (scripts) που απαιτούνται για τη λειτουργία ενός εξυπηρετητή καθώς και όλους τους ορισμούς των dataBlocks των αντικειμένων που υπάρχουν στην εφαρμογή. Ο φάκελος του client περιέχει τα αρχεία που απαιτούνται για τη λειτουργία του πελάτη. Τα αρχεία αυτά είναι λιγότερα από τα αντίστοιχα του εξυπηρετητή καθώς όταν ο πελάτης συνδέεται με έναν εξυπηρετητή φορτώνει από αυτόν όλα τα dataBlocks που χρειάζονται. Επίσης, σε ξεχωριστό φάκελο (client/ui) περιέχει τα αρχεία που έχουν να κάνουν με το γραφικό περιβάλλον (με κατάληξη .gui) καθώς και τις αντίστοιχες εικόνες που καλούν αυτά τα αρχεία. Τέλος, ο φάκελος data περιέχει όλα τα απαραίτητα textures των αντικειμένων, τους ήχους, τις εικόνες, τα μοντέλα και φυσικά το πολύ σημαντικό αρχείο της αποστολής. Παρακάτω θα παρουσιάσουμε πιο αναλυτικά τη δομή και τις βασικές λειτουργίες κάθε φακέλου και αρχείου.

- Η δομή του φακέλου **client** παρουσιάζεται σχηματικά παρακάτω:



Εικόνα 4.5: Δομή του φακέλου client

Το αρχείο init.cs περιέχει όλες τις απαραίτητες αρχικοποιήσεις για να ξεκινήσει η εφαρμογή καθώς καλεί όλα τα αρχεία που πρέπει να εκτελεστούν και φορτώνει το βασικό μενού του παιχνιδιού.
Το αρχείο config.cs ορίζει τη λειτουργία κάθε πλήκτρου και πώς επιδρά το πάτημά του στην ροή της εφαρμογής.

Το αρχείο prefs.cs περιέχει τις επιθυμητές ρυθμίσεις για τη βέλτιστη εκτέλεση του παιχνιδιού. Οι ρυθμίσεις αυτές περιλαμβάνουν τη λειτουργία της οθόνης, του ποντικιού, του πληκτρολογίου ή joystick εφόσον αυτό υπάρχει.
Τέλος το αρχείο defaults.cs περιλαμβάνει επιθυμητές ρυθμίσεις που είναι προεπιλεγμένες στο παιχνίδι και αφορούν την εικόνα και τον ήχο.

Ο φάκελος ***scripts*** περιλαμβάνει με τη σειρά του τα παρακάτω αρχεία:

- audioProfiles.cs
- centerPrint.cs
- chatHud.cs
- client.cs
- default.bind.cs
- game.cs
- loadingGui.cs
- messageHud.cs
- missionDownload.cs
- optionsDlg.cs
- playerList.cs
- playGui.cs
- serverConnections.cs

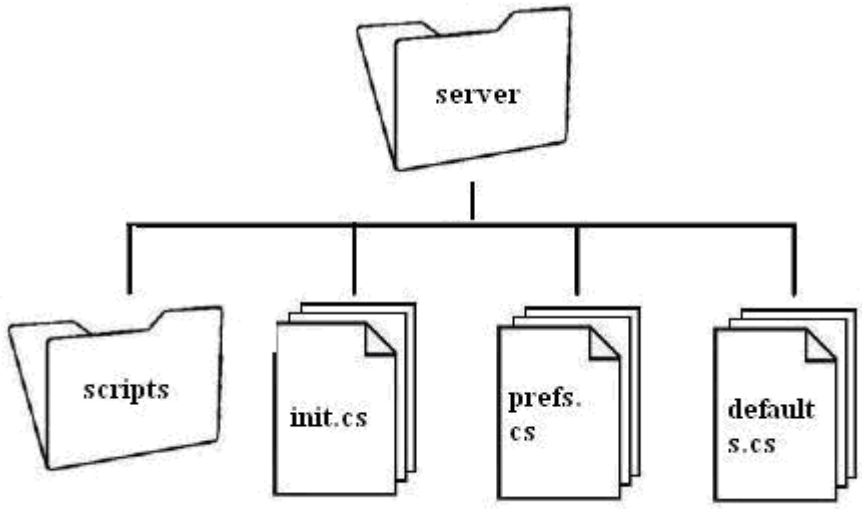
Θα ήταν υπερβολή να αναλυθεί λεπτομερώς η λειτουργία καθενός από τα παραπάνω αρχεία αλλά συνολικά μπορούμε να πούμε ότι είναι υπεύθυνα για την απρόσκοπτη σύνδεση του πελάτη σε κάποιον εξυπηρετητή και την ‘κληρονόμηση’ από αυτόν όλων των απαραίτητων ρυθμίσεων και φυσικά των dataBlocks.

Ο φάκελος ***ui*** περιέχει τα αρχεία για κάθε διαφορετικό γραφικό περιβάλλον που μπορεί να φορτωθεί στην αποστολή καθώς επίσης και τις απαραίτητες εικόνες που απαιτεί αυτή η διαδικασία. Στο συγκεκριμένο φάκελο βρίσκουμε ενδεικτικά τα αρχεία:

- StartupGui.gui
- gameOverGui.gui
- startMissionGui.gui
- youWinGui.gui
- playGui.gui
- mainMenuGui.gui

τα οποία περιγράφουν τη δομή του γραφικού περιβάλλοντος της έναρξης, της ήττας, της αρχής της αποστολής, της νίκης, του κυρίως μέρους τη αποστολής καθώς και του κεντρικού μενού από το οποίο ο χρήστης κάνει διάφορες επιλογές όπως για παράδειγμα να συνεχίσει ή να εγκαταλείψει την εφαρμογή.

- Η δομή του φακέλου **server** παρουσιάζεται σχηματικά παρακάτω:



Εικόνα 4.6: Δομή του φακέλου server

Τα αρχεία init.cs, prefs.cs, defaults.cs έχουν τις ίδιες λειτουργίες με τα αντίστοιχα του φακέλου client που περιγράψαμε παραπάνω.

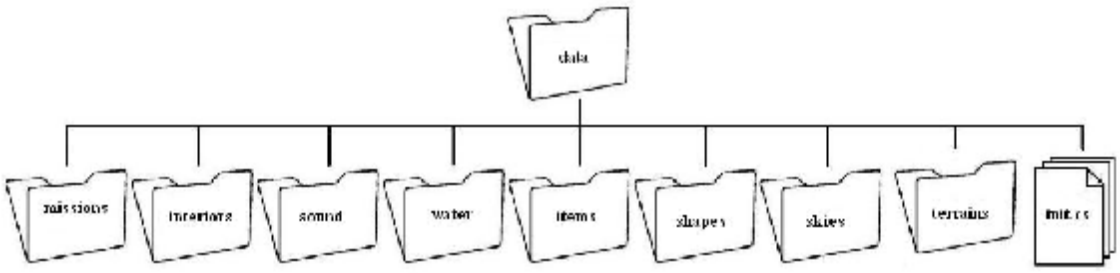
Ο φάκελος *scripts* σε αυτή την περίπτωση περιέχει περισσότερα αρχεία τα οποία παρουσιάζονται ονομαστικά στη συνέχεια:

- audioProfiles.cs
- camera.cs
- car.cs
- centerPrint.cs
- checkpoint.cs
- commands.cs
- game.cs
- markers.cs
- radiusDamage.cs
- shapeBase.cs
- staticShape.cs
- triggers.cs
- healthkit.cs
- narki.cs

Στα συγκεκριμένα αρχεία περιέχονται όλα τα dataBlocks καθώς και όλες οι συναρτήσεις και οι ρυθμίσεις που καθορίζουν τη ροή του παιχνιδιού. Ανάλυση αυτών των συναρτήσεων σύμφωνα με τις οποίες εκτελείται η εφαρμογή θα γίνει στην επόμενη παράγραφο που αναφέρεται στους κανόνες που διέπουν το συγκεκριμένο ηλεκτρονικό παιχνίδι.

Ο τελευταίος φάκελος που πρέπει να εξετάσουμε για να έχουμε μία ολοκληρωμένη εικόνα για το πώς διαρθρώνεται και οργανώνεται η εφαρμογή μας, είναι ο φάκελος data των δεδομένων.

- Η σχηματική αναπαράσταση του φακέλου **data** φαίνεται στη συνέχεια:



Εικόνα 4.7: Δομή του φακέλου data

Βλέπουμε ότι και σε αυτόν το φάκελο συναντάμε το αρχείο αρχικοποίησης init.cs, όπως και στις δύο προηγούμενες περιπτώσεις, το οποίο όμως αυτή τη φορά περιλαμβάνει μόνο μία αρχικοποίηση που αφορά την προτεραιότητα εμφάνισης των αντικειμένων στην οθόνη του παιχνιδιού. Οι υπόλοιποι φάκελοι περιλαμβάνουν τα εξής:

- **Missions:** Περιλαμβάνει το αρχείο της αποστολής racing.mis στο οποίο ορίζεται κάθε αντικείμενο που συμμετέχει στην εφαρμογή. Υπάρχουν επίσης και κάποια βοηθητικά αρχεία που αφορούν τη μορφολογία του περιβάλλοντος (πχ έδαφος) του παιχνιδιού.
- **Interiors:** Περιλαμβάνει κατά κύριο λόγο τις εικόνες του εφαρμόζονται στα ‘εσωτερικά’ αντικείμενα της αποστολής, όπως μπάρες, πινακίδες και εμπόδια.
- **Sound:** Σε αυτό τον φάκελο περιέχονται όλα τα αρχεία ήχου που χρησιμοποιούνται στην εφαρμογή.
- **Water:** Αυτός ο φάκελος περιέχει διάφορα textures που μπορούν να χρησιμοποιηθούν για την απεικόνιση του νερού. Η συγκεκριμένη αποστολή που υλοποιήθηκε δεν περιλαμβάνει το στοιχείο του νερού.
- **Items:** Εδώ βρίσκουμε τις εικόνες που αποτελούν την υφή (texture) των αντικειμένων τύπου item που χρησιμοποιήθηκαν στην εφαρμογή. Σε αυτά τα αντικείμενα θα αναφερθούμε λεπτομερώς στη συνέχεια.
- **Shapes:** Ο συγκεκριμένος φάκελος περιέχει όλα τα μοντέλα των αντικειμένων που εμπλέκονται στην εφαρμογή όπως τα δέντρα, το όχημα, οι νάρκες και άλλα. Επίσης συμπεριλαμβάνεται και μία ειδική κατηγορία ‘αντικειμένων’ που αποτελούνται από σωματίδια (particles) και μπορούν να χρησιμοποιηθούν για την αναπαράσταση κάποιων εφέ όπως είναι αυτό της έκρηξης.
- **Skies:** Σε αυτό το σημείο βρίσκουμε διάφορα textures κατάλληλα για την απεικόνιση του ουρανού.
- **Terrains:** Τέλος, σε αυτό το φάκελο τοποθετούνται εικόνες και μοντέλα που έχουν να κάνουν με την αναπαράσταση του εδάφους.

Έχοντας πλέον μία πλήρη εικόνα της διάρθρωσης της εφαρμογής θα προχωρήσουμε στην επεξήγηση του τρόπου λειτουργίας της.

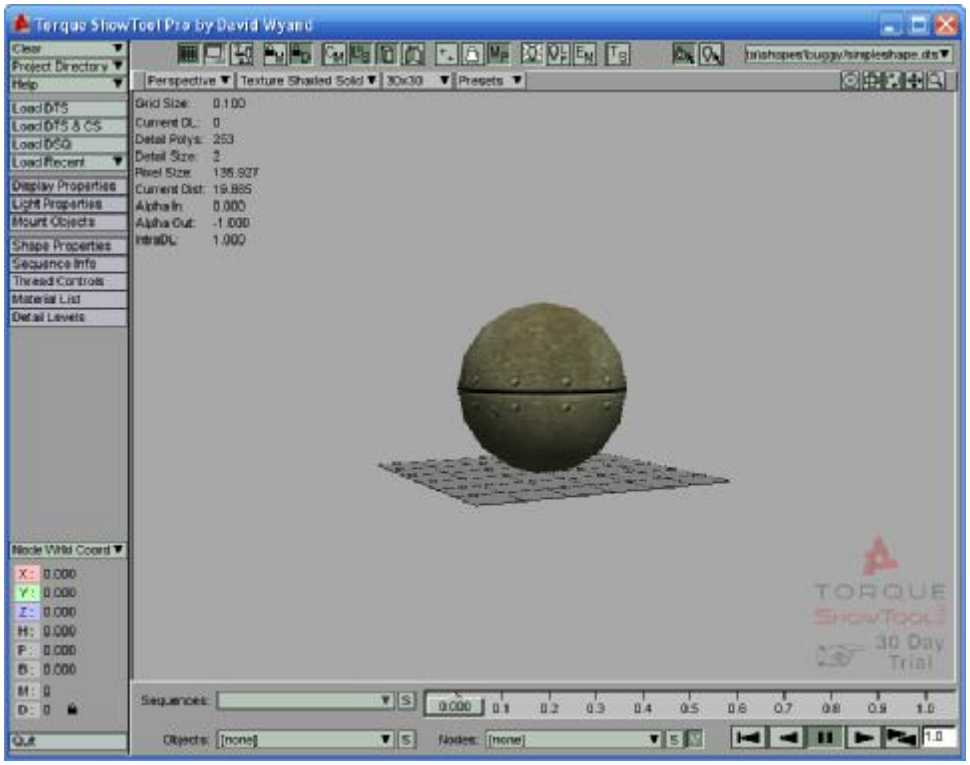
4.5 Κανόνες του ηλεκρονικού παιχνιδιού

Το πρώτο βήμα για την κατασκευή του ηλεκτρονικού μας παιχνιδιού ήταν η δημιουργία των κανόνων ώστε να έχει ενδιαφέρον για τον παίχτη και να τον κρατάει σε εγρήγορση ώστε να αντιλαμβάνεται τις αλλαγές που θα προκύπτουν. Οι κανόνες αυτοί είναι οι εξής:
Ο παίχτης μπορεί να οδηγήσει το όχημα και να κινηθεί προς κάθε κατεύθυνση, με τη βοήθεια των κατάλληλων πλήκτρων πλοήγησης, στο γραφικό περιβάλλον του παιχνιδιού.

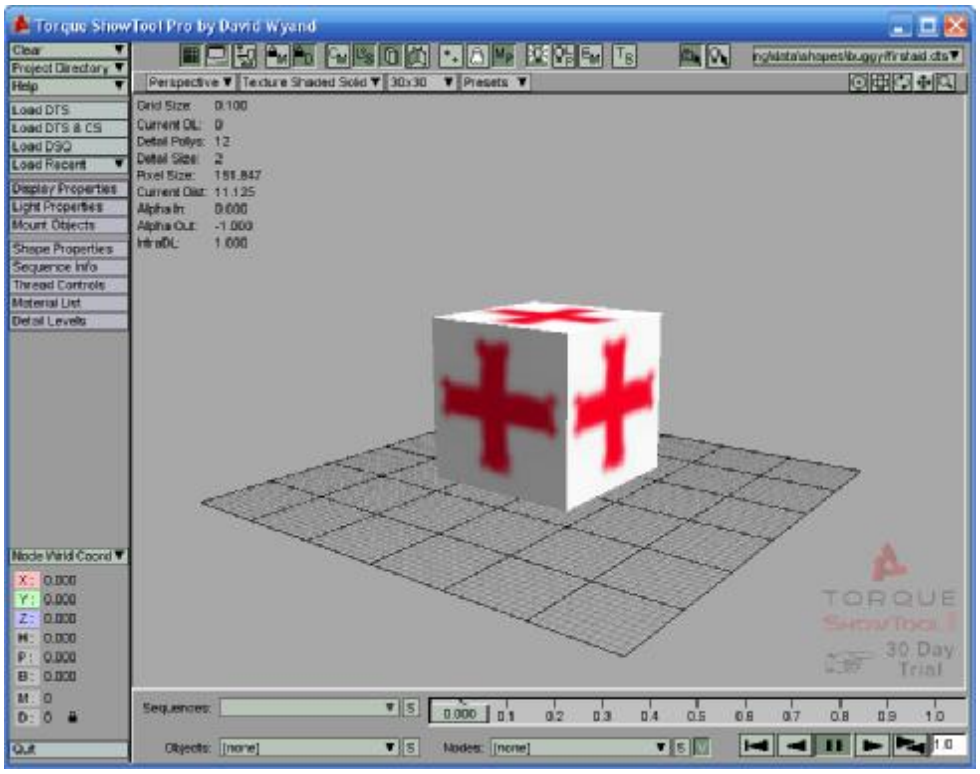
Εικόνα 4.8: Το γραφικό περιβάλλον του παιχνιδιού

Το γραφικό περιβάλλον αποτελείται από αντικείμενα αλληλεπιδραστικά και μη αλληλεπιδραστικά. Τα μη αλληλεπιδραστικά είναι τα δέντρα και οι θάμνοι ενώ τα αλληλεπιδραστικά που έχουν και ουσιαστικό ρόλο στην εξέλιξη του παιχνιδιού είναι οι νάρκες (mines) και τα ‘κουτιά ζωής’ (healthkits).
Πριν αναλυθεί η λειτουργία αυτών των αντικειμένων στο παιχνίδι θα γίνει μία σύντομη αναφορά στον τρόπο δημιουργίας τους.
Και τα δύο αυτά αντικείμενα δημιουργήθηκαν με τη βοήθεια του προγράμματος 3D Studio Max το οποίο χρησιμοποιείται για την κατασκευή τρισδιάστατων στατικών και δυναμικών μοντέλων. Ως textures των μοντέλων χρησιμοποιήθηκαν κατάλληλες PNG

εικόνες οι οποίες κάλυψαν τα σχήματα που δημιουργήσαμε και τους έδωσαν την επιθυμητή υφή. Με τη βοήθεια του εργαλείου Torque ShowTool παρουσιάζεται η μορφή των αντικειμένων που δημιουργήθηκαν με το 3D Studio Max προκειμένου να εισαχθούν στο περιβάλλον της εφαρμογής μας:

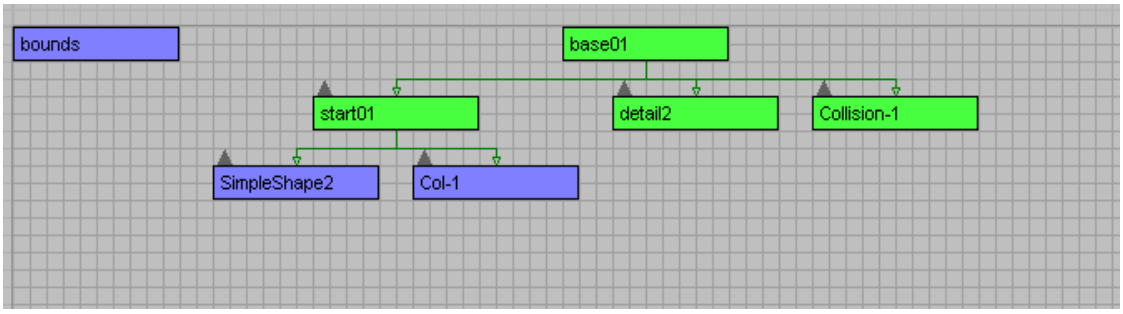


Εικόνα 4.9: Επισκόπιση του αντικειμένου της νάρκης



Εικόνα 4.10: Επισκόπιση του αντικειμένου Healthkit

Καθώς τα μοντέλα του 3D Studio Max σώζονται σαν .max αρχεία ενώ τα αρχεία που χρησιμοποιεί το Torque Game Engine είναι .dts, είναι προφανές ότι απαιτείται ένα πρόγραμμα-plugin για το 3DSMax που θα επιτρέπει την αλλαγή του αρχείου σε .dts. Το πρόγραμμα αυτό είναι το Max2DTS και χρειάζεται τις ακόλουθες ρυθμίσεις από το χρήστη για να λειτουργήσει:
Αρχικά απαιτείται μια ιεραρχία από dummy objects: Το base01 είναι η ρίζα με παιδιά το detailX όπου X το επίπεδο λεπτομέρειας που μας ενδιαφέρει (το μοντέλο μπορεί να έχει πολλά επίπεδα λεπτομέρειας, έτσι ώστε όταν είναι μακριά να αποκτά ένα απλούστερο σχήμα για να μειώνει τις απαιτήσεις από τον υπολογιστή), το collision-1 το οποίο καταδεικνύει ότι υπάρχει collision mesh με αριθμό 1 και τέλος το start01 στο οποίο προσαρτώνται τα μοντέλα (το col-1 που είναι το collision mesh και το nameX όπου name είναι το όνομα του αντικειμένου και X το επίπεδο λεπτομέρειας). Τέλος υπάρχει ένα ορθογώνιο μοντέλο bounds το οποίο περικλείει τα δύο μοντέλα (αντικείμενο και collision mesh) και το οποίο δεν προσαρτάται σε κανένα dummy object. Αυτό το αντικείμενο είναι το ‘περίβλημα’ του μοντέλου, και είναι απαραίτητο για όλα τα DTS μοντέλα.
Η ιεραρχία που περιγράφηκε, όπως προκύπτει από το 3D Studio Max, είναι σχηματικά η ακόλουθη:



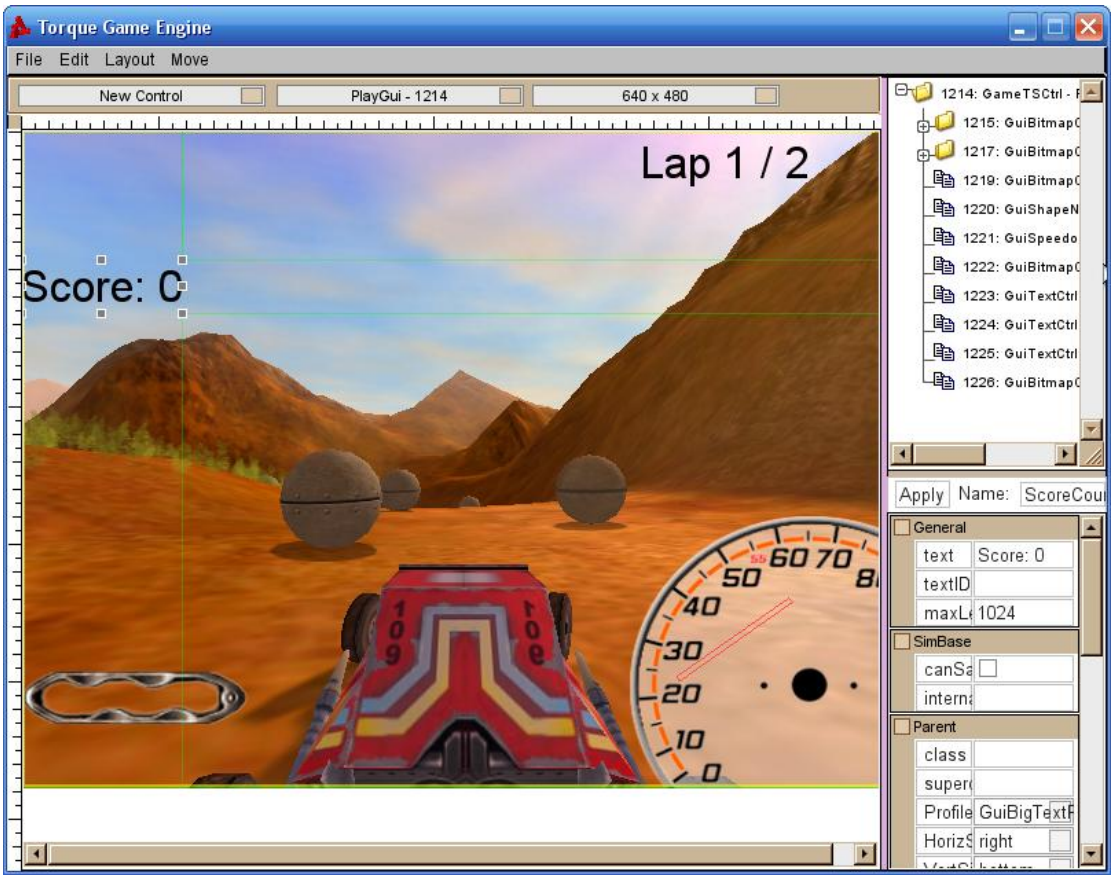
Εικόνα 4.11: Ιεραρχία αντικειμένων από το 3D Studio Max

Το collision mesh είναι στην ουσία η 'σκληρή επιφάνεια' του μοντέλου, με την οποία μπορεί ο χρήστης να συγκρουστεί. Ένα αντικείμενο χωρίς collision mesh επιτρέπει στον χρήστη να περάσει από μέσα του, ενώ παράλληλα ο χρήστης δεν μπορεί να συγκρουστεί μαζί του. Τα αντικείμενα πρέπει να έχουν ένα προσεγμένο collision mesh αλλά παράλληλα και απλό έτσι ώστε να μειώνονται οι απαιτήσεις υπολογιστικής ισχύος.

Πέρα από τα αντικείμενα που περιγράψαμε παραπάνω, προστέθηκαν και δύο άλλες παράμετροι στην εφαρμογή μας που αφορούν το score του παίχτη και το επίπεδο της ζωής του. Η πρώτη παράμετρος ελέγχεται από τη μεταβλητή score και εμφανίζεται αρχικά στην οθόνη με το κείμενο: **Score: 0**. Στο αρχείο playGui.gui που περιλαμβάνει όλα τα στοιχεία της επιφάνειας του γραφικού περιβάλλοντος (GUI – Graphic User Interface) η παράμετρος του score ορίζεται ως εξής:

```
new GuiTextCtrl (ScoreCounter) {
    canSaveDynamicFields = "0";
    Profile = "GuiBigTextProfile"; //Είδος της γραμματοσειράς που χρησιμοποιείται
    HorizSizing = "right"; // Θέση ως προς τον οριζόντιο άξονα
    VertSizing = "bottom"; // Θέση ως προς τον κάθετο άξονα
    position = "0 95";
    Extent = "118 40";
    MinExtent = "8 2";
    canSave = "1";
    Visible = "1";
    hovertime = "1000";
    text = "Score: 0";
    maxLength = "1024";
  };
```

Σε αυτό το τμήμα του κώδικα, ορίζεται η θέση του κειμένου που θα εμφανίζει το score την οθόνη, το είδος της γραμματοσειράς που θα χρησιμοποιηθεί για την απεικόνισή του καθώς και το συνολικό μήκος του string.



Εικόνα 4.12: Ρύθμιση των παραμέτρων του score από τον GUI editor

Το score φυσικά δε θα μπορούσε να είναι κάτι στατικό. Έτσι στην εφαρμογή μας το score αυξάνεται κατά 1 κάθε φορά που ο χρήστης συγκρούεται με ένα healthkit. Στο περιβάλλον της εφαρμογής υπάρχουν διασκορπισμένα σε διάφορες θέσεις 10 healthkits και μόλις ο χρήστης φτάσει σε score 10 (δηλαδή καταφέρει να συγκρουστεί και με τα 10 healthkits χωρίς όμως να χάσει τη ζωή του όπως θα αναλύσουμε στη συνέχεια), κερδίζει και εμφανίζεται το ανάλογο μήνυμα στην οθόνη.
Παρακάτω φαίνεται η δομή της συνάρτησης της σύγκρουσης του παίχτη και το αντικείμενο healthkit:

```
function healthkit::onCollision(%this, %obj, %col)
{
  %col_className = %col.getClassname();
  %col_dblock_className =%col.getClassName();
  %colName = %col.getDataBlock().getName;
//Ελεγχος εάν το αντικείμενο που συγκρούεται με το helathkit είναι το όχημα του χρήστη
  if (%col.getClassName() $= "WheeledVehicle")
        {
//Κατά τη σύγκρουση με το αντικείμενο healthkit ακούγεται και κατάλληλος ήχος
          alxPlay(bonus);
          %client=%col.client;
```

```
//Αύξηση του score μετά τη σύγκρουση
        %client.score++;
//Εντολές για ανάλογη συμπεριφορά προς τον πελάτη
        commandToClient(%client,'SetScoreCounter', %client.score);
        commandToClient(%client, 'PlayCollision2');
//Μετά τη σύγκρουση το αντικείμενο healthkit διαγράφεται από το γραφικό περιβάλλον
//του παιχνιδιού.
        %obj.delete();
//Ελεγχος εάν ο χρήστης έχει φτάσει σε score 10 να εμφανιστεί το κατάλληλο μήνυμα
//νίκης στην οθόνη.
                if (%client.score ==10)
                {
                        loadyouWinGui();
                }
        }
}
```

Το συγκεκριμένο τμήμα κώδικα που βρίσκεται στο αρχείο server/scripts/healthkit.cs περιγράφει αυτό που αναλύθηκε παραπάνω. Συγκεκριμένα: η onCollision() ελέγχει εάν το αντικείμενο που συγκρούστηκε με το healthkit είναι τύπου 'WheeledVehicle', δηλαδή το όχημα του παίχτη μας. Στην περίπτωση που υπήρξε πράγματι σύγκρουση helathkit – παίχτη δίνεται η εντολή μέσω της συνάρτησης alxPlay() να ακουστεί ένας ήχος που έχουμε ορίσει και έχει ονομαστεί bonus. Στη συνέχεια αυξάνεται το score κατά ένα μέσω της εντολής:

%client.score++;

Η εντολή CommandToClient που ακολουθεί είναι μία από τις σημαντικότερες αυτής της εφαρμογής και θα αναλυθεί εκτενώς στο επόμενο κεφάλαιο που αφορά τους τρόπους παρέμβασης στη ροή του παιχνιδιού. Προς στο παρόν είναι αρκετό να αναφερθεί ότι με αυτό τον τρόπο δίνει εντολή ο server στον client να κάνει μία ενέργεια. Στη συγκεκριμένη περίπτωση με την:

commandToClient(%client,'SetScoreCounter', %client.score);

δίνεται η εντολή στον client να μεταβάλλει και αυτός την τιμή του score με τον ίδιο τρόπο που μεταβλήθηκε και στο server. Η εντολή commandToClient(%client, '*func*', arg1, …, angn) συνοδεύεται πάντα από μία συνάρτηση ClientCommand*func* στη μεριά του client η οποία θα κληθεί να εκτελέσει την εντολή του server. Έτσι, στο αρχείο client/scripts/game.cs βρίσκουμε τον κώδικα:

```
function clientCmdSetScoreCounter(%score)
{
        ScoreCounter.setText("Score:" SPC %score);
}
```

σύμφωνα με τον οποίο ρυθμίζεται ανάλογα με το server και το score του client.

Με ανάλογο τρόπο λειτουργεί και η εντολή:

commandToClient(%client, 'PlayCollision2');

η οποία ακολουθείται από τη συνάρτηση:

```
function clientCmdPlayCollision2(%obj)
{
        alxPlay(bonus);

}
```

στο client/scripts/game.cs, σύμφωνα με την οποία δίνεται εντολή στον client να αναπαράξει τον ήχο bonus που ακούγεται έπειτα από σύγκρουση με healthkit. Να τονιστεί ότι αυτή η διαδικασία είναι απαραίτητη για την αναπαραγωγή των ήχων στην πλευρά του client καθώς αυτοί δεν 'κληρονομούνται' από το server.
Στη συνέχεια, το αντικείμενο healthkit διαγράφεται από το γραφικό περιβάλλον του παιχνιδιού μέσω της εντολής:

%obj.delete();

Η εικόνα του παιχνιδιού μετά από μία σύγκρουση με healthkit (το score έχει αυξηθεί κατά 1) είναι η ακόλουθη:

Εικόνα 4.13: Αύξηση του score μετά από σύγκρουση με helathkit

Το τελικό στάδιο της συνάρτησης onCollision() είναι ο έλεγχος του score, και όταν αυτό φτάσει την τιμή 10, φορτώνεται κατάλληλο γραφικό περιβάλλον νίκης το οποίο εμφανίζει το παρακάτω μήνυμα επιτυχίας:

Εικόνα 4.14: Μήνυμα που εμφανίζεται σε περίπτωση νίκης του παίχτη

Το συγκεκριμένο γραφικό περιβάλλον περιγράφεται στο αρχείο client/ui/youWin.gui και έχει τον ακόλουθο κώδικα:

```
new GuiFadeinBitmapCtrl(youWinGui) {
   profile = "GuiInputCtrlProfile";
   horizSizing = "right";
   vertSizing = "bottom";
   position = "0 0";
   extent = "640 480";
   minExtent = "8 8";
   visible = "1";
   helpTag = "0";
   bitmap = "./youWin";
   wrap = "0";
   fadeinTime = "125";
   waitTime    = "5000"; //Ορίζεται ο χρόνος παραμονής του μηνύματος στην οθόνη του
//χρήστη
   fadeoutTime = "125";
};
//--- OBJECT WRITE END ---

function loadyouWinGui()
{
```

```
   StartupGui.done = false;
   Canvas.setContent( youWinGui );
   schedule(100, 0, checkyouWinDone );

//Αναπαραγωγή κατάλληλου ήχου νίκης
   alxPlay(Victory);
}

//-------------------------------------
function youWinGui::click()
{
   youWinGui.done = true;
}

//-------------------------------------
// Έλεγχος νίκης
function checkyouWinDone()
{

//Εφόσον εμφανιστεί το μήνημα νίκης, μετά την παραμονή του στην οθόνη για τον χρόνο
//που έχει οριστεί (5 sec) θα φορτωθεί το βασικό μενού για να επιλέξει ο χρήστης αν θα
//συνεχίσει ή αν θα εγκαταλέιψει.
   if (youWinGui.done)
   {
            echo ("**Load Main Menu**");
      loadMainMenu();
   }
   else
      schedule(100, 0, checkyouWinDone );
}
```

Σύμφωνα με αυτό το αρχείο, σε περίπτωση νίκης εμφανίζεται στο χρήστη η εικόνα 4.14 που φαίνεται παραπάνω, ακούγεται ο ήχος ‘victory’ που έχει οριστεί και στη συνέχεια επιστρέφουμε στο βασικό μενού όπου μπορεί να επιλεγεί συνέχιση ή μη του παιχνιδιού ανάλογα με τη θέληση του χρήστη.

Εικόνα 4.15: Επιλογές βασικού μενού

Η δεύτερη παράμετρος ελέγχεται από τη μεταβλητή health (ζωή) και εμφανίζεται αρχικά στην οθόνη με το κείμενο: **Health: 100**. Έχουμε δηλαδή αρχικοποιήσει τη μεταβλητή στην τιμή 100. Στο αρχείο client/ui/playGui.gui η παράμετρος της ζωής ορίζεται ως εξής:

```
//Ορισμός των παραμέτρων που αφορούν τη ζωή, όπως: θέση, είδος γραμματοσειράς,
//μήκος του string
new GuiTextCtrl(HealthCounter) {
     canSaveDynamicFields = "0";
     Profile = "HealthPrintProfile";
     HorizSizing = "right";
     VertSizing = "bottom";
     position = "40 523";
     Extent = "77 20";
     MinExtent = "8 8";
     canSave = "1";
     Visible = "1";
     hovertime = "1000";
     text = "Health: 100";
     maxLength = "255";
   };
```

```
//Ορισμός των παραμέτρων του πλαισίου το οποίο θα περικλείει την τιμή που θα ορίζει //το επίπεδο της ζωής του παίχτη που του απομένει
   new GuiBitmapCtrl() {
      canSaveDynamicFields = "0";
      Profile = "GuiDefaultProfile";
      HorizSizing = "right";
      VertSizing = "top";
      position = "1 395";
      Extent = "164 38";
      MinExtent = "8 8";
      canSave = "1";
      Visible = "1";
      hovertime = "1000";
      bitmap = "./healthBar";
      wrap = "0";
   };
};
```

Σε αυτό το τμήμα κώδικα ορίζεται το σημείο της οθόνης στο οποίο θα εμφανίζεται η τιμή της ζωής του παίχτη καθώς επίσης και το είδος της γραμματοσειράς, το μήκος του string και το αρχικό κείμενο που θα εμφανίζεται κατά την έναρξη του παιχνιδιού.

Η τιμή της ζωής είναι κάτι μεταβλητό και μεταβάλλεται με τον εξής τρόπο: Κάθε φορά που ο παίχτης συγκρούεται με ένα αντικείμενο της κλάσης narki χάνεται το 20% της ζωής. Συνεπώς αν συγκρουστεί πέντε φορές τότε φορτώνεται το αντίστοιχο γραφικό περιβάλλον ήττας και ο χρήστης χάνει. Παράλληλα, τη στιγμή της σύγκρουσης ακούγεται κατάλληλος ήχος που έχει οριστεί και δημιουργείται ένα ειδικό εφέ έκρηξης, συστατικά τα οποία είναι απαραίτητα για να είναι το παιχνίδι πιο ενδιαφέρον και αλληλεπιδραστικό ως προς το χρήστη. Τα ‘συστατικά’ της έκρηξης ορίζονται στο αρχείο server/scripts/narki.cs και παρουσιάζονται στη συνέχεια:

```
//Ορίζονται οι παράμετροι για την εμφάνιση σπίθας
datablock ParticleData(narkiExplosionSparks)
{
   textureName          = "~/data/shapes/particles/spark";
   dragCoefficient      = 1;
   gravityCoefficient   = 0.0;
   inheritedVelFactor   = 0.2;
   constantAcceleration = 0.0;
   lifetimeMS           = 500;
   lifetimeVarianceMS   = 350;

   colors[0]     = "0.60 0.40 0.30 1.0";
   colors[1]     = "0.60 0.40 0.30 1.0";
   colors[2]     = "1.0 0.40 0.30 0.0";
```

```
   sizes[0]      = 0.25;
   sizes[1]      = 0.15;
   sizes[2]      = 0.15;

   times[0]      = 0.0;
   times[1]      = 0.5;
   times[2]      = 1.0;
};

//Απαραίτητο dataBlock υπεύθυνο για την 'εκτόξευση' της σπίθας  που ορίστηκε παραπάνω
datablock ParticleEmitterData(narkiExplosionSparkEmitter)
{
   ejectionPeriodMS = 3;
   periodVarianceMS = 0;
   ejectionVelocity = 5;
   velocityVariance = 1;
   ejectionOffset   = 0.0;
   thetaMin         = 0;
   thetaMax         = 180;
   phiReferenceVel  = 0;
   phiVariance      = 360;
   overrideAdvances = false;
   orientParticles  = true;
   lifetimeMS       = 100;
   particles = "narkiExplosionSparks";
};

//Ορισμός των παραμέτρων του συστατικού της φωτιάς
datablock ParticleData(narkiExplosionFire)
{
   textureName          = "~/data/shapes/particles/fire";
   dragCoeffiecient     = 100.0;
   gravityCoefficient   = 0;
   inheritedVelFactor   = 0.25;
   constantAcceleration = 0.1;
   lifetimeMS           = 1200;
   lifetimeVarianceMS   = 300;
   useInvAlpha =  false;
   spinRandomMin = -80.0;
   spinRandomMax =  80.0;

   colors[0]     = "0.8 0.4 0 0.8";
   colors[1]     = "0.2 0.0 0 0.8";
   colors[2]     = "0.0 0.0 0.0 0.0";

   sizes[0]      = 1.5;
```

```
   sizes[1]      = 0.9;
   sizes[2]      = 0.5;

   times[0]      = 0.0;
   times[1]      = 0.5;
   times[2]      = 1.0;
};
//Απαραίτητο dataBlock υπεύθυνο για την 'εκτόξευση' της φωτιάς  που ορίστηκε
//παραπάνω
datablock ParticleEmitterData(narkiExplosionFireEmitter)
{
   ejectionPeriodMS = 10;
   periodVarianceMS = 0;
   ejectionVelocity = 0.8;
   velocityVariance = 0.5;
   thetaMin         = 0.0;
   thetaMax         = 180.0;
   lifetimeMS       = 250;
   particles = "narkiExplosionFire";
};

//Ορίζονται οι παράμετροι του συστατικού του καπνού
datablock ParticleData(narkiExplosionSmoke)
{
   textureName          = "~/data/shapes/particles/smoke";
   dragCoeffiecient     = 100.0;
   gravityCoefficient   = 0;
   inheritedVelFactor   = 0.25;
   constantAcceleration = -0.30;
   lifetimeMS           = 1200;
   lifetimeVarianceMS   = 300;
   useInvAlpha =  true;
   spinRandomMin = -80.0;
   spinRandomMax =  80.0;

   colors[0]     = "0.56 0.36 0.26 1.0";
   colors[1]     = "0.2 0.2 0.2 1.0";
   colors[2]     = "0.0 0.0 0.0 0.0";

   sizes[0]      = 4.0;
   sizes[1]      = 2.5;
   sizes[2]      = 1.0;

   times[0]      = 0.0;
   times[1]      = 0.5;
   times[2]      = 1.0;
```

```
};

//Απαραίτητο dataBlock υπεύθυνο για την 'εκτόξευση' του καπνού που ορίστηκε παραπάνω
datablock ParticleEmitterData(narkiExplosionSmokeEmitter)
{
   ejectionPeriodMS = 10;
   periodVarianceMS = 0;
   ejectionVelocity = 4;
   velocityVariance = 0.5;
   thetaMin         = 0.0;
   thetaMax         = 180.0;
   lifetimeMS       = 250;
   particles = "narkiExplosionSmoke";
};

//Σύνθεση των παραπάνω συστατικών σε δύο κατηγορίες
datablock ExplosionData(narkiSubExplosion1)
{
   offset = 0;
   emitter[0] = narkiExplosionSmokeEmitter;
   emitter[1] = narkiExplosionSparkEmitter;
};

datablock ExplosionData(narkiSubExplosion2)
{
   offset = 1.0;
   emitter[0] = narkiExplosionSmokeEmitter;
   emitter[1] = narkiExplosionSparkEmitter;
};

//Σύνθεση των παραπάνω δύο κατηγοριών στο τελικό εφέ της έκρηξης
datablock ExplosionData(narkiExplosion)
{

   particleEmitter = narkiExplosionFireEmitter;
   particleDensity = 75;
   particleRadius = 2;
   soundProfile =AudioExplosion; //Ορισμός του ηχητικού εφέ κατά τη διάρκεια της
//σύγκρουσης

   emitter[0] = narkiExplosionSmokeEmitter;
   emitter[1] = narkiExplosionSparkEmitter;

   subExplosion[0] = narkiSubExplosion1;
   subExplosion[1] = narkiSubExplosion2;
```

```
// Ορίζουμε να υπαρξει μετακίνηση της κάμερας για πιο ρεαλιστικό αποτέλεσμα
   shakeCamera = true;
   camShakeFreq = "10.0 11.0 10.0";
   camShakeAmp = "1.0 1.0 1.0";
   camShakeDuration = 0.5;
   camShakeRadius = 10.0;

// 'Ωστική δύναμη' της έκρηξης
   impulseRadius = 10;
   impulseForce = 15;

//Λάμψη
   lightStartRadius = 6;
   lightEndRadius = 3;
   lightStartColor = "0.5 0.5 0";
   lightEndColor = "0 0 0";
};

//Ορισμός του αντικειμένου της νάρκης και του τύπου της  έκρηξής της
datablock StaticShapeData (narki)
{
        shapeFile = "~/data/shapes/buggy/simpleshape.dts";
        category = "Items";
        mass=1;
        friction=1;
        emap = true;
        damageAmount =20;
        explosion = narkiExplosion;
};
```

Όπως φαίνεται και από τον παραπάνω κώδικα, για τη δημιουργία μίας έκρηξης, ορίζουμε κάποια dataBlocks τύπου ParticleData και κάποια άλλα τύπου ParticleEmmiterData. Τα πρώτα ορίζουν αυτό που θα μπορούσαμε να χαρακτηρίσουμε ως 'υφή' της έκρηξης και τα δεύτερα χαρακτηρίζουν τον τρόπο που αυτή η 'υφή' λειτουργεί και γίνεται ένα δυναμικό εφέ. Μεταβάλλοντας κάποιες από τις παραπάνω παραμέτρους αλλάζει και η μορφή της έκρηξης και μπορεί να αποκτήσει μεγαλύτερη ή μικρότερη διάρκεια, διαφορετική έκταση, ένταση και λάμψη.
Στη συνέχεια, ακολουθεί ο κώδικας που περιγράφει τη διαδικασία σύγκρουσης του παίχτη με νάρκη που περιγράφεται στο αρχείο server/scripts/game.cs:

```
//Συνάρτηση σύγκρουσης νάρκης με το όχημα του παίχτη

function narki::onCollision( %this, %obj, %col )
{
        if( %col.getType() & $TypeMasks::ShapeBaseObjectType )
            {
```

```
                    %col.damage( %obj, %obj.getPosition(), 50, "Mine" );
                    %obj.setDamageState(Destroyed);
                    %obj.schedule(200, "delete");
                    %client=%col.client;

//Μείωση της ζωής κατά 20%
                    %client.Health =%client.Health-20;

//Εντολές προς τον πελάτη για μείωση της ζωής και αναπαραγωγή του ήχου
                    commandToClient(%client, 'PlayCollision', %obj);
                    commandToClient(%client,'SetHealthCounter', %client.Health);

//Ελεγχος εάν έχει μηδενιστεί η ζωή ώστε αν φορτωθεί το κατάλληλο γραφικό περιβάλλον
//ήττας
                  if (%client.Health = =0)
                           {
                               loadgameOverGui();
        for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ )
{
            %cl = ClientGroup.getObject( %clientIndex );
            commandToClient(%cl, 'loadgameOverGui');
}
                      }
        }
}

function narki::onDestroyed(%data, %obj, %pos, %mod)
{
        %pos = %obj.getPosition();
        radiusDamage( %obj, %pos, 4.0 ,%data.radiusDamage, "MineRadius",
%data.areaImpulse);
}
```

Αυτό που κάνει η συγκεκριμένη συνάρτηση είναι να ελέγχει εάν το αντικείμενο της σύγκρουσης ανήκει σε μία απο τις κατηγορίες Items ή StaticShapes και εφ'όσον ανήκει να του εφαρμόζει το εφέ της έκρηξης και να το διαγράφει από το γραφικό περιβάλλον του παιχνιδιού.
Στη συνέχεια, η πορεία εξέλιξης του παιχνιδιού ορίζει τη μείωση της ζωής του χρήστη κατά 20% μέσω της εντολής:

%client.Health =%client.Health-20;

ενώ ακολουθούν οι εντολές:

commandToClient(%client,'SetHealthCounter', %client.Health);
commandToClient(%client, 'PlayCollision');

οι οποίες αναγκάζουν τον πελάτη να μεταβάλει και αυτός με τη σειρά του την τιμή της ζωής του και να αναπαράξει τον ήχο της συγκρούσης. Όπως έχουμε ήδη αναφέρει οι εντολές commandToClient(%client, '*func*', arg1, …, angn) συνοδεύονται πάντα από συναρτήσεις ClientCommand*func* στη μεριά του client. Έτσι και για τις παραπάνω εντολές βρίσκουμε στο αρχείο client/game.cs τα παρακάτω κομμάτια κώδικα:

```
function clientCmdSetHealthCounter(%Health)
{
	HealthCounter.setText("Health:" SPC %Health);
}

function clientCmdPlayCollision(%obj)
{
	alxPlay(AudioExplosion);
}
```

Η εικόνα 4.16 παρουσιάζει το αποτέλεσμα όσων περιγράψαμε παραπάνω όταν κατά τη διάρκεια του παιχνιδιού συμβεί σύγκρουση μεταξύ του οχήματος του χρήστη και μίας νάρκης:

Εικόνα 4.16: Έκρηξη και μείωση της ζωής σύγκρουση

Τέλος, συνεχίζοντας την ανάλυση της συνάρτησης σύγκρουσης, βλέπουμε ότι γίνεται ελέγχος για το αν η ζωή του χρήστη έχει μηδενιστεί και σε περίπτωση που έχει συμβεί κάτι τέτοιο εκτελείται η εντολή:

loadgameOverGui();

η οποία αναφέρεται στο αρχείο client/ui/gameOver.gui:

```
new GuiFadeinBitmapCtrl(gameOverGui) {
   profile = "GuiInputCtrlProfile";
   horizSizing = "right";
   vertSizing = "bottom";
   position = "0 0";
   extent = "640 480";
   minExtent = "8 8";
   visible = "1";
   helpTag = "0";
   bitmap = "./gameOver";
   wrap = "0";
   fadeinTime = "125";
   waitTime   = "3000";
   fadeoutTime = "125";
};
//--- OBJECT WRITE END ---

function loadgameOverGui()
{
   StartupGui.done = false;
   Canvas.setContent( gameOverGui );
   schedule(100, 0, checkStartupDone );

   alxPlay(gameOver);
}

//-------------------------------------
function gameOverGui::click()
{
   gameOverGui.done = true;
}

//-------------------------------------
function checkgameOverDone()
{
   if (gameOver.done)
   {
            echo ("Load Main Menu");
```

```
    loadStartup();
  }
  else
    schedule(100, 0, checkgameOver );
}
```

Σύμφωνα με αυτό το αρχείο, σε περίπτωση ήττας εμφανίζεται στο χρήστη κατάλληλο μήνυμα στην οθόνη που φαίνεται παρακάτω, ενώ παράλληλα ακούγεται και κατάλληλο ηχητικό εφέ που ορίσαμε.

Εικόνα 4.17: Μήνυμα ήττας

Τέλος, με την προσθήκη κατάλληλων αρχείων στους φακέλους engine/platform, engine/sim, engine/platform32 και κάποιες τροποποιήσεις στα αρχεία clinet/prefs.cs και client/scripts/default.bind.cs καταφέραμε να συνδέσουμε τιμόνι και πεντάλ οδήγησης έτσι ώστε η εφαρμογή να είναι πιο δελεστική και αλληλεπιδραστική για το χρήστη.

Με γνωστούς τώρα όλους του κανόνες που διέπουν το συγκεκριμένο ηλεκρονικό παιχνίδι, μπορούμε να προχωρήσουμε στην ανάλυση των τρόπων παρέμβασης στη φυσιολογική του ροή και στα αποτελέσματα που προκύπτουν κατά την αξιολόγησή του.

5 ΤΡΟΠΟΙ ΠΑΡΕΜΒΑΣΗΣ ΣΤΗ ΡΟΗ ΤΟΥ ΠΑΙΧΝΙΔΙΟΥ

5.1 Εισαγωγή

Όπως έχει ήδη αναφερθεί από την αρχή της αναφοράς, σκοπός της συγκεκριμένης εφαρμογής είναι η μελέτη της σχέσης που υπάρχει μεταξύ της συναισθηματικής κατάστασης του χρήστη και των εκφράσων του προσώπου του ώστε να μπορέσουμε να προσθέσουμε γνώση στον τομέα της συναισθηματικής αλληλεπίδρασης ανθρώπου-υπολογιστή. Στο κεφάλαιο που ακολουθεί θα παρουσιαστούν οι μέθοδοι που εφαρμόσαμε και μας βοήθησαν στην επίτευξη αυτού του στόχου.

5.2 Torque και δικτυακές εφαρμογές

Η βάση πάνω στην οποία στηρίζεται η εφαρμογή που αναπτύχθηκε είναι η δυνατότητα του Torque να υποστηρίζει multiplayer παιχνίδια, δηλαδή παιχνίδια στα οποία μπορούν να συμμετέχουν πολλοί παίχτες εκ των οποίων ο ένας είναι ο εξυπηρετητής (server) και όλοι οι υπόλοιποι είναι οι πελάτες (clients). Άλλωστε, όπως έχει ήδη αναφερθεί στο κεφάλαιο 3, ένα από τα βασικότερα πλεονεκτήματα του Torque είναι η δυνατότητα που παρέχει στο χρήστη για networking (δικτυακές) εφαρμογές. Παρακάτω φαίνεται η δυνατότητα επιλογής του χρήστη σχετικά με το αν επιθυμεί παιχνίδι πολλαπλών παικτών ή όχι:

Εικόνα 5.1: Δυνατότητα επιλογής του χρήστη για παιχνίδι με πολλαπλούς χρήστες

Σε περίπτωση που επιλεγεί από το χρήστη η επιλογή ‘Host Multiplayer’ μπορούν στη συνέχεια να συνδεθούν και άλλοι χρήστες σε αυτό τον αρχικό τον οποίο θα αντιμετωπίζουν ως εξυπηρετητή. Απαραίτητη προϋπόθεση για να συμβεί αυτό είναι οι υπολογιστές και των υπόλοιπων χρηστών να είναι συνδεδεμένοι σε δίκτυο με τον αρχικό. Σε αυτή την περίπτωση θα πρέπει στο αρχικό μενού ο χρήστης να επιλέξει τη σύνδεση με το server όπως φαίνεται παρακάτω:

Εικόνα 5.2: Επιλογή του χρήστη να συνδεθεί μέσω δικτύου με κάποιον εξυπηρετητή

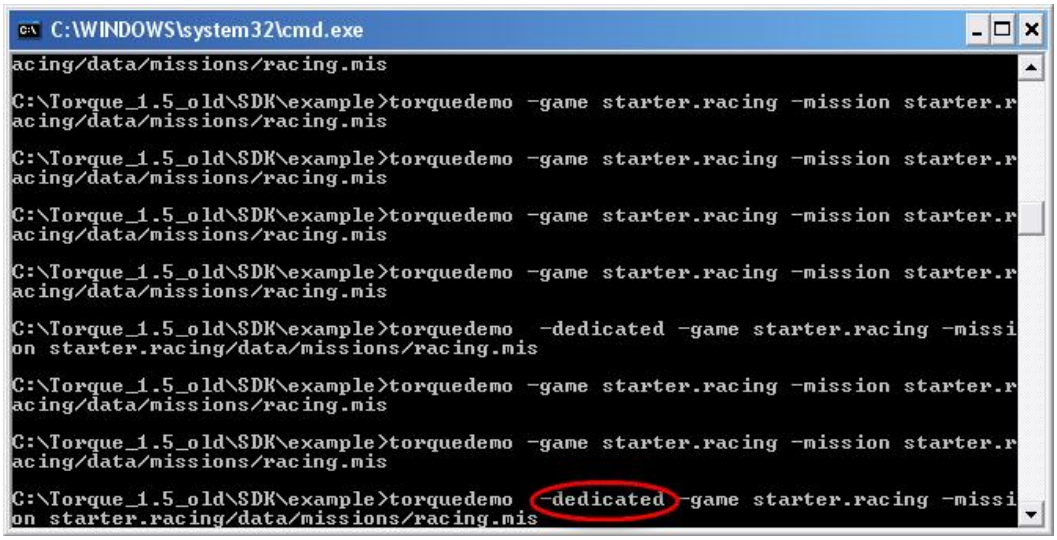
Στη συνέχεια θα επιλεγεί η αναζήτηση δικτύου και τέλος ο εξυπηρετητής στον οποίο ο χρήστης επιθυμεί να συνδεθεί:

Εικόνα 5.3: Επιλογή του επιθυμητού εξυπηρετητή μετά την αναζήτηση δικτύου (Query Lan)

Σε αυτή την περίπτωση, θα υπάρξουν στο γραφικό περιβάλλον του παιχνιδιού τόσα οχήματα όσοι είναι και οι πελάτες που συνδέθηκαν με τον εξυπηρετητή συν το όχημα του τελευταίου. Παρακάτω δίνεται η εικόνα του παιχνιδιού, από τον υπολογιστή του εξυπηρετητή, όταν υπάρχουν δύο χρήστες εκ των οποίων ο χρήστης ‘athina’ είναι ο εξυπηρετητής και ο χρήστης ‘athina.1’ ο πελάτης:

Εικόνα 5.4: Ύπαρξη πολλαπλών χρηστών στην ίδια αποστολή

Όπως έχει ήδη αναφερθεί, στη συγκεκριμένη εφαρμογή επιθυμούμε να μελετήσουμε τις αυθόρμητες αντιδράσεις του χρήστη και συνεπώς δε θα ήταν χρήσιμο να γνωρίζει και την ύπαρξη ενός άλλου παίχτη που μπορεί να περεμβαίνει στη ροή του παιχνιδιού. Συνεπώς η δομή πελάτη-εξυπηρετητή όπου καθένας εκ των δύο διαθέτει το δικό του όχημα στο γραφικό περιβάλλον του παιχνιδιού δεν είναι βολική.
Έτσι, για να κατορθώσουμε να παρεμβαίνουμε στη φυσιολογική ροή της εφαρμογής μας χωρίς ο χρήστης να καταλαβαίνει κάτι τέτοιο, χρησιμοποιήσαμε έναν 'αφοσιωμένο εξυπηρετητή' (dedicated server) τον οποίο θέτουμε σε εφαρμογή από τη γραμμή εντολών γράφοντας τα ακόλουθα:



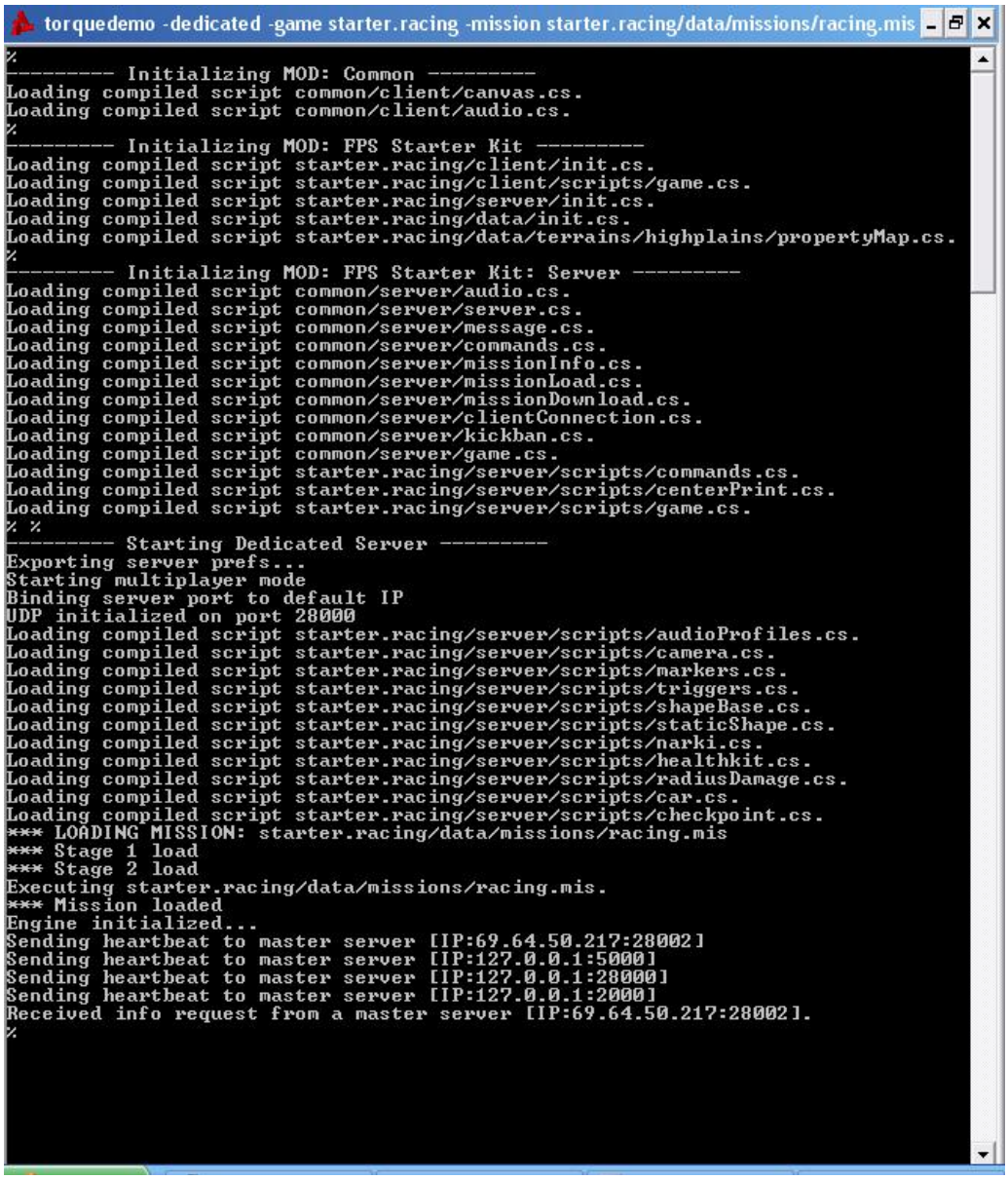
Εικόνα 5.5: Διαδικασία κλήσης ενός αφοσιωμένου εξυπηρετητή

Η ύπαρξη των συγκεκριμένων ορισμάτων προκειμένου να ξεκινήσει το παιχνίδι και η αποστολή racing.mis, υπό τη μορφή αφοσιωμένου εξυπηρετητή ορίζεται στο αρχείο main.cs της εφαρμογής με τον παρακάτω κώδικα:

```
package StarterRacing {

function displayHelp() {
  Parent::displayHelp();
  error(
    "FPS Mod options:\n"@
    "  -dedicated                Start as dedicated server\n"@
    "  -connect <address>     For non-dedicated: Connect to a game at <address>\n" @
    "  -mission <filename>    For dedicated: Load the mission\n"
  );
}

function parseArgs()
{
  Parent::parseArgs();

  // Arguments, which override everything else.
  for (%i = 1; %i < $Game::argc ; %i++)
  {
    %arg = $Game::argv[%i];
    %nextArg = $Game::argv[%i+1];
    %hasNextArg = $Game::argc - %i > 1;

    switch$ (%arg)
    {
```

```
      case "-dedicated":
        $Server::Dedicated = true;
        enableWinConsole(true);
        $argUsed[%i]++;

      //--------------------
      case "-mission":
        $argUsed[%i]++;
        if (%hasNextArg) {
          $missionArg = %nextArg;
          $argUsed[%i+1]++;
          %i++;
        }
        else
          error("Error: Missing Command Line argument. Usage: -mission <filename>");
    }
  }
}
```

Η μορφή του αφοσιωμένου εξυπηρετητή είναι αυτή που φαίνεται παρακάτω:



Εικόνα 5.6: Ο αφοσιωμένος εξυπηρετητής

Ο συγκεκριμένος εξυπηρετητής κάνει ό,τι και ο απλός εξυπηρετητής με τη διαφορά ότι δε διαθέτει γραφικό περιβάλλον παρά μόνο την παραπάνω κονσόλα από την οποία μπορεί να ελέγχει το παιχνίδι χωρίς ο πελάτης να καταλαβάινει την παρουσία του. Ο τρόπος που θα εκμεταλλευτούμε αυτό τον αφοσιωμένο εξυπηρετητή και οι συναρτήσεις που θα χρησιμοποιήσουμε θα αναλυθούν σε επόμενη παράγραφο. Πριν όμως, θα ήταν χρήσιμο να αναλύσουμε λίγο πιο διεξοδικά τη διαδικασία και τα γεγονότα που συμβαίνουν όταν έχουμε τη σύνδεση πελάτη-εξυπηρετητή.

5.3 Διαδιακασία σύνδεσης πελάτη σε εξυπηρετητή

Όταν ένας πελάτης συνδέεται σε έναν εξυπηρετητή, λαμβάνουν χώρα διάφορα γεγονότα τα οποία ολοκληρώνονται σε τρεις ξεχωριστές φάσεις:
Κατά την πρώτη φάση φορτώνονται τα dataBlocks, κατά τη δεύτερη φορτώνονται τα στιγμιότυπα των αντικειμένων (Ghost Objects) και κατά τη τρίτη φωτίζεται η σκηνή. Για να γίνει περισσότερο κατανοητός ο τρόπος που επικοινωνούν και μοιράζονται δεδομένα ο πελάτης και ο εξυπηρετητής θα παραθέσουμε όλα τα γεγονότα που διαδραματίζονται ακολουθώντας τη χρονική τους αλληλουχία:

Client	Server
Αρχικά κάνει ο πελάτης αίτηση σύνδεσης για έναρξη της αποστολής.	Αποδοχή αίτησης και έναρξη της πρώτης φάσης της αποστολής σύμφωνα με το αρχείο: *common/server/scripts/missionDownload.cs* στο οποίο βρίσκουμε την εντολή έναρξης: *commandToClient('MissionStartPhase1')* Διαβάζουμε: "*** Sending mission load to client: <missionInfo>"
Ενεργοποιείται από το αρχείο */common/client/missionDownload.cs* ο παρακάτω handler: *clientCmdMissionStartPhase1()* Στη συνέχεια εμφανίζονται τα παρακάτω μηνύματα: • "*** New Mission: <MissionName>" • *** Phase 1: Download Datablocks & Targets" Ακολούθως καλείται από το αρχείο *starter.racing/client/scripts/missionDownload.cs* η συνάρτηση *onMissionDownloadPhase1()* η οποία εκτελεί τις ακόλουθες ενέργειες: • Κλείνει όσα παράθυρα διαλόγου είναι ανοιχτά. • Αρχικοποιεί το γραφικό περιβάλλον με την *LoadingProgressGui* 'γεμίζοντας σταδικά τη μπάρα' που γράφει	Ενεργοποιείται από το αρχείο */common/server/missionDownload.cs* ο παρακάτω handler: *serverCmdMissionStartPhase1Ack()* Στη συνέχεια λαμβάνουν χώρα τα ακόλουθα γεγονότα: • Αρχίζει η φάση της μεταφοράς των DataBlocks όπως ορίζεται στο *engine/game/gameConnection.cc--ConsoleMethod(GameConnection, transmitDataBlocks...)* • Η φάση αυτή συνεχίζεται μέχρι να σταλούν όλα τα DataBlocks οπότε και αποστέλλεται μήνυμα στον client που τον ενημερώνει ότι η διαδικασία τελείωσε. • Αποστέλλεται ένα δικτυακό γεγονός (NetEvent) για κάθε DataBlock που απομένει προς αποστολή στο DataBlockGroup

"LOADING DATABLOCKS". Αυτή η ενέργεια μας δείχνει την εξέλιξη της διαδικασία φόρτωσης των DataBlocks. • Εκτελείται η εντολή *commandToServer('MissionStartPhase1Ack')*	
• Ο πελάτης λαμβάνει το δικτυακό γεγονός που συνοδεύει κάθε DataBlock όπως ορίζεται στο: *engine/platform/GameInterface::postEvent()* • Ο αντίστοιχος handler ενεργοποιείται και λαμβάνει το γεγονός και στη συνέχεια επεξεργάζεται κάθε DataBlock όπως πρέπει. Η διαδικασία αυτή περιγράφεται στο: */engine/platform/gameInterface.cc--GameInterface::processEvent()*	Μόλις ολοκληρωθεί η αποστολή όλων των DataBlocks αποστέλλεται δικτυακό γεγονός που να το υποδηλώνει: *NetEvent: DataBlocksDone*
Επιβεβαιώνει ο client ότι έχει λάβει όλα τα DataBlocks και στέλνει το ακόλουθο δικτυακό γεγονός πίσω στο server: *NetEvent: DataBlocksDownloadDone*	Ο server επεξεργάζεται το γεγονός *DataBlocksDownloadDone* που έλαβε από τον client και καλεί τη συνάρτηση: *GameConnection::onDataBlocksDone* από το αρχείο: *common/server/missionDownload.cs* Στη συνέχεια εκτελείται η εντολή: *commandToClient('MissionStartPhase2');* προκειμένου να ξεκινήσει η δεύτερη φάση της σύνδεσης.
• Λαμβάνεται το δικτυακό γεγονός *MissionStartPhase2* και ενεργοποιείται ο handler *clientCmdMissionStartPhase2()* που βρίσκεται στο αρχείο: *common/client/missionDownload.cs* • Καλείται η συνάρτηση που υποδηλώνει την ολοκλήρωση της πρώτης φάσης *onPhase1Complete()* από το	Ενεργοποιείται ο handler *serverCmdMissionStartPhase2Ack()* από το αρχείο *common/server/missionDownload.cs* • Αρχίζει η αντιγραφή των αντικειμένων καλώντας *την Console Method activateGhosting* • Καλείται από το αρχείο *engine/sim/netGhost.cc* η *NetConnection::activateGhosting()* • Αποστέλλεται μήνυμα σύνδεσης

αρχείο: *starter.racing/client/scripts/missionDownload.cs* • Στη συνέχεια αρχίζει η προετοιμασία της δεύτερης φάσης όπου θα φορτωθούν τα αντικείμενα της αποστολής και διαβάζουμε: "*** Phase 2: Download Ghost Objects". • Καλείται η συνάρτηση *onMissionDownloadPhase2()* από το αρχείο: *starter.racing/client/scripts/missionDownload.cs* και αρχικοποιεί το νέο γραφικό περιβάλλον με τη *LoadingProgressGui* γεμίζοντας σταδικά τη μπάρα που γράφει LOADING OBJECTS. Αυτή η ενέργεια μας δείχνει την εξέλιξη της διαδικασία φόρτωσης των αντικειμένων. • Εκτελείται η *commandToServer('MissionStartPhase2Ack')*.	*connectionMessage: GhostAlwaysStarting* • Ο server αποστέλλει το δικτυακό γεγονός *GhostAlwaysObjectEvent* για κάθε αντικείμενο
Επεξεργασία του δικτυακού γεγονότος: *GhostAlwaysObjectEvent*	Αποστολή μηνύματος σύνδεσης *connectionMessage: GhostAlwaysDone*
Χειρισμός του μηνύματος σύνδεσης *connectionMessage:GhostAlwaysDone* όπως ορίζεται στο: *engine/sim/netGhost.cc--NetConnection::handleConnectionMessage()*	Συντονισμός των ghostAlways αντικειμένων που πρέπει να αποσταλλούν πολλαπλά.
Μετά τα πολλαπλής αποστολής αντικείμενα αποστέλλεται μήνυμα *sendConnectionMessage(ReadyForNormalGhosts)* το οποίο υποδηλώνει ότι ο πελάτης είναι έτοιμος να λάβει τα απλά αντικείμενα.	Λήψη του μηνύματος *ReadyForNormalGhosts* και εκτέλεση της συνάρτησης: *GameConnection::onGhostAlwaysObjectsReceived* από το αρχείο *common/server/missionDownload.cs* Στη συνέχεια εκτελείται η εντολή *commandToClient('MissionStartPhase3')* προκειμένου να ξεκινήσει η τρίτη φάση της

	σύνδεσης.
• Ενεργοποίηση του handler *clientCmdMissionStartPhase3()* από το αρχείο: *common/client/missionDownload.cs* • Κλήση της *onPhase2Complete()* από το αρχείο *starter.racing/client/scripts/missionDownload.cs* που δηλώνει την ολοκλήρωση της δεύτερης φάσης. • Ξεκινάει η επιβεβαίωση των αντικειμένων καλώντας την *ConsoleMethod(StartClientReplication)* από το αρχείο: *engine/game/fx/fxShapeReplicator.cc* • Γίνεται η επιβεαίωση των αντικειμένων και διαβάζουμε: "Client Replication Startup has Happened!" • Ξεκινάει η 'φωλιασμένη' επιβεβαίωση με κλήση της *ConsoleFunction(StartFoliageReplication)* από το αρχείο *engine/game/fx/fxFoliageReplicator.cc* • Εκτελείται η 'φωλιασμένη' επιβεβαίωση και διαβάζουμε: "fxFoliageReplicator - Client Foliage Replication Startup is complete." • Στη συνέχεια διαβάζουμε: "*** Phase 3: Mission Lighting" • Γίνεται η έναρξη της τρίτης φάσης που συνίσταται στο φωτισμό της αποστολής.mission lighting • Διαβάζουμε: "Lighting mission...." • Φορτώνεται το γραφικό περιβάλλον που δείχνει τη διαδικασία εξέλιξης της αποστολής του φωτισμού.	• Ενεργοποίηση του handler *serverCmdMissionStartPhase3Ack()* από το αρχείο *common/server/scripts/missionDownload.cs* • Στη συνέχεια εκτελείται η συνάρτηση *GameConnection::startMission()* του αρχείου *common/server/clientConnection* και η εντολή *commandToClient('MissionStart')* προκειμένου να ξεκινήσει η αποστολή στον πελάτη.

Καλείται η *onMissionDownloadPhase3* για να αρχικοποιηθεί το γραφικό περιβάλλον. • Διαβάζουμε: "Mission lighting done" • Εφόσον ολοκληρώνεται και η τρίτη φάση καλείται η αντίστοιχη συνάρτηση *onPhase3Complete()* από το αρχείο *starter.racing/client/scripts/missionDownload.cs* και τέλος από το ίδιο αρχείο εκτελείται η συνάρτηση *onMissionDownloadComplete()* που δηλώνει ότι η διαδικασία σύνδεσης τελείωσε. • Εκτελείται η εντολή *commandToServer('MissionStartPhase3Ack')*	
Ενεργοποίηση του handler *clientCmdMissionStart()* στο αρχείο *common/client/scripts/mission.cs*	• Σε αυτό το σημείο έχει ολοκληρωθεί η μεταφορά όλων των απαραίτητων στοιχείων και μπορεί να συνδεθεί ο client στο server με τον τρόπο που ορίζεται από τη συνάρτηση *GameConnection::onClientEnterGame()* στο αρχείο *common/server/scripts/game.cs* η οποία ορίζεται σε script στο αρχείο: *starter.racing/server/scripts/game.cs* μέσω της συνάρτησης *GameConnection::onClientEnterGame()* που θα παραθέσουμε στη συνέχεια. • Τέλος, συγχρονίζεται η ώρα σε client και server • Δημιουργείται το αντικείμενο της κάμερας • Αποφασίζεται το σημείο εμφάνισης που ορίζεται από τις SpawnSpheres και δημιουργείται το όχημα (γενικά Avatar) του παίχτη.

Μετά από αυτή τη σειρά των ενεργειών ο πελάτης μπορεί να συνδεθεί στον εξυπηρετητή και να ξεκινήσει το παιχνίδι σύμφωνα με τις ρυθμίσεις που έχουν γίνει στη συνάρτηση

GameConnection::onClientEnterGame(%this). Η συνάρτηση αυτή που βρίσκουμε στο αρχείο server/scripts/game.cs παρατίθεται στη συνέχεια:

```
function GameConnection::onClientEnterGame(%this)
{
   commandToClient(%this, 'SyncClock', $Sim::Time - $Game::StartTime);
   commandToClient(%this, 'SetMaxLaps', $Game::Laps);

// Εμφανίζεται στην οθόνη αντίστροφη μέτρηση 3 2 1 και τελικά η λέξη GO για να
//ξεκινήσει το παιχνίδι.
   for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ ) {
      %cl = ClientGroup.getObject( %clientIndex );

            schedule(1000, 0, "countThree"); //Η συνάρτηση 'countThree' είναι υπεύθυνη
//για την εμφάνιση της αντίστροφης μέτρησης

            }

//Το παιχνίδι υποστηρίζει λειτουργία FirstPerson όπου ο παίχτης βρίσκεται ουσιαστικά
//μέσα στο όχημα και λειτουργία ThirdPerson όπου βλέπει το όχημα που οδηγεί εξωτερικά.
//Εδώ έχουμε θέσει ως προεπιλογή για τον πελάτη τη δεύτερη περίπτωση καθώς παρέχει
//καλύτερη εποπτεία του χώρου. Η αλλαγή από τη μία λειτουργία στην άλλη μπορεί να
//γίνει με το πάτημα του πλήκτρου tab
   %this.setFirstPerson(false);

//Δημιουργία του αντικειμένου της κάμερας
   %this.camera = new Camera() {
      dataBlock = Observer;
   };
   MissionCleanup.add( %this.camera );
   %this.camera.scopeToClient(%this);

//Όμοια με πριν θέτουμε ως προεπιλογή και για τον εξυπηρετητή στην περίπτωση που
//παίζει χωρίς πελάτη συνδεδεμένο τη λειτουργία ThirdPerson, με την κάμερα δηλαδή στο
//εξωτερικό του οχήματος.
   ServerConnection.setFirstPerson(false);

      %this.setControlObject(%this.camera);

      %this.spawnCar();

//με την έναρξη του παιχνιδιού του πελάτη ορίζουμε να έχει έλεγχο του οχήματος
                  %this.setControlObject(%this.car);
                  %this.camera.setOrbitMode(%this.car, %this.car.getTransform(), 0.5, 4.5,
4.5);
```

```
}
```

Για λόγους πληρότητας θα παραθέσουμε και τον κώδικα της συνάρτησης ‘countThree’ που συναντήσαμε παραπάνω:

```
function countThree()
{
   for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ ) {
      %cl = ClientGroup.getObject( %clientIndex );
//Δίνεται εντολή στον πελάτη να εμφανίσει στην οθόνη του το ‘3’ για την αντίστροφη
//μέτρηση. Φυσικά αυτή η εντολή συνοδεύεται από μια συνάρτηση handler
//clientCmdPlayCounter() στο αρχείο client/scripts/game.cs που πραγματοποιεί αυτή την
//εντολή.
      commandToClient(%cl, 'PlayCounter', 3);
   }
//Με αυτή την εντολή προγραμματίζεται η έναρξη εκτέλεσης της συνάρτησης ‘countTwo’ σε
//ένα δευτερόλεπτο.
            schedule(1000, 0, "countTwo");
}

function countTwo()
{
 for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ ) {
      %cl = ClientGroup.getObject( %clientIndex );
//Δίνεται εντολή στον πελάτη να εμφανίσει στην οθόνη του το ‘2’ για την αντίστροφη
//μέτρηση. Φυσικά αυτή η εντολή συνοδεύεται από τη συνάρτηση handler
//clientCmdPlayCounter() στο αρχείο client/scripts/game.cs που πραγματοποιεί αυτή την
//εντολή.
   commandToClient(%cl, 'PlayCounter', 2);
   }
//Με αυτή την εντολή προγραμματίζεται η έναρξη εκτέλεσης της συνάρτησης ‘countOne’ σε
//ένα δευτερόλεπτο.
           schedule(1000, 0, "countOne");
}

function countOne()
{
      for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ ) {
      %cl = ClientGroup.getObject( %clientIndex );
//Δίνεται εντολή στον πελάτη να εμφανίσει στην οθόνη του το ‘1’ για την αντίστροφη
//μέτρηση. Φυσικά αυτή η εντολή συνοδεύεται από τη συνάρτηση handler
//clientCmdPlayCounter() στο αρχείο client/scripts/game.cs που πραγματοποιεί αυτή την
//εντολή.

      commandToClient(%cl, 'PlayCounter', 1);
   }
```

```
//Προγραμματίζεται η έναρξη εκτέλεσης της συνάρτησης 'startRace' σε ένα δευτερόλεπτο.
        schedule(1000, 0, "startRace");
}

function startRace()
{
  for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ ) {
    %cl = ClientGroup.getObject( %clientIndex );
    commandToClient(%cl, 'PlayStart');
  }

    %cl.lap = 0;
    %cl.nextCheck = 1;
    %cl.setControlObject(%cl.car);
    %cl.camera.setFlyMode();

    $Game::Running = true;

}
```

Σε αυτό το σημείο θα ήταν χρήσιμο να αποσαφηνίσουμε κάποιο στοιχείο που μέχρι τώρα δεν έχει αναφερθεί σαφώς: Όταν ξεκινάει η εφαρμογή σε κάποιο υπολογιστή δίνεται η δυνατότητα σε χρήστες άλλων τερματικών που είναι συνδεδεμένοι σε τοπικό δίκτυο με τον αρχικό μας υπολογιστή, να συνδεθούν σε αυτόν ως clients θεωρώντας τον αρχικό ως server. Αυτός όμως ο server είναι απλά ένας τοπικός server ο οποίος ξεκινώντας την εφαρμογή συνδέεται ως client με τον κεντρικό server της GarageGames. Άλλωστε και στον αφοσιωμένο server που καλούμε για τις ανάγκες του πειράματός μας, διαβάζουμε ανά τακτά χρονικά διαστήματα τη φράση:

Sending heartbeat to master server [IP:127.0.0.1:28000]

όπου η συγκεκριμένη διεύθυνση ορίζεται στις αρχικοποιήσεις του παιχνιδιού. Συνεπώς κάθε φορά που ξεκινάμε τη λειτουργία ενός server αυτός θα ακολουθήσει όλη τη διαδικασία των τριών φάσεων που προαναφέραμε για να φορτώσει την αποστολή του από τον κεντρικό server και τελικά θα προσπελάσει και θα εκτελέσει και τη συνάρτηση GameConnection::onClientEnterGame(). Εάν στη συνέχεια κάποιος άλλος χρήστης συνδεθεί στον πρώτο ως πελάτης θα ακολουθήσει την ίδια διαδικασία αλλά στη θέση του server της GarageGames θα είναι ο τοπικός server που ξεκινήσαμε προηγουμένως.

Τελειώνοντας με αυτή την ανάλυση, μπορούμε πλέον να προχωρήσουμε στον τρόπο και στις συναρτήσεις που χρησιμοποιήθηκαν προκειμένου να επικοινωνήσει ο αφοσιωμένος εξυπηρετητής του πειράματός μας με τον πελάτη-χρήστη.

5.4 Χρήσιμες συναρτήσεις

Η βασικές συναρτήσεις στις οποίες βασιστήκαμε για την ανάπτυξη αυτής της εφαρμογής έχουν ήδη αναφερθεί και παραπάνω αλλα στο συγκεκριμένο σημείο θα αναλυθούν εκτενέστερα. Πρόκειται για τις commandToClient() και commandToServer(). Για την ακρίβεια η δεύτερη εντολή δε χρησιμοποιήθηκε αλλά θα αναφερθεί απλώς για λόγους πληρότητας ως προς την επικοινωνία πελάτη-εξυπηρετητή.
Η σύνταξη των παραπάνω εντολών είναι η ακόλουθη:

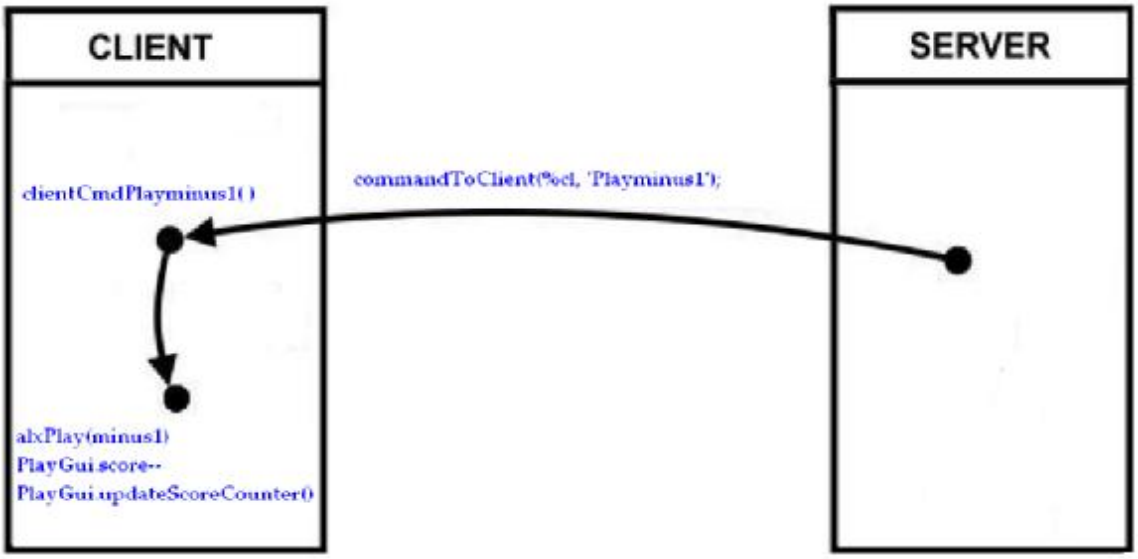
commandToClient (client, func[arg1, …, angn]);

commadToServer (func[arg1, …, angn]);

Η πρώτη αφορά εντολή που δίνει ο εξυπηρετητής σε κάποιον πελάτη (client) που θα ορίσει ο ίδιος, για να εκτελέσει τη συνάρτηση func με ορίσματα τα arg1, …, angn. Η commandToServer(), αφορά εντολή που δίνει κάποιος πελάτης στον εξυπηρετητή προκειμένου να εκτελεστεί στον τελευταίο η συνάρτηση func[arg1, …, angn]). Και οι δύο εντολές λειτουργούν με παρόμοιο τρόπο και στη συνέχεια θα αναφερθούμε στην commandToClient() ενώ η λειτουργία της commandToServer() θα είναι πλήρως ανάλογη.
Όταν εκτελείται η εντολή commandToClient() στον εξυπηρετητή, ενεργοποιείται στην πλευρά του πελάτη ένας ‘handler’ που θα χειριστεί αυτή τη συνάρτηση. Η μορφή που πρέπει να έχει ο handler είναι η ακόλουθη:

clientCmd*func*();

όπου στη θέση της func είναι το ίδιο όνομα της συνάρτησης που περιέχει ως όρισμα η commandToClient().
Στη συνέχεια θα παραθέσουμε ένα ενδεικτικό σχηματικό διάγραμμα όπου φαίνεται η λειτουργία της εντολής CommandToClient() σε μία από τις συναρτήσεις που χρησιμοποιήσαμε και θα παρουσιαστούν παρακάτω και συγκεκριμένα στην ‘serverdecreasescore’:



Εικόνα 5.7: Λειτουργία της εντολής CommandToClient()

5.5 Συναρτήσεις παρεμβολής

Προκειμένου να παρέμβουμε στην κανονική ροή του παιχνιδιού δημιουργήσαμε συναρτήσεις παρεμβολής που προκαλούν κάποια απρόοπτα γεγονότα. Αυτές είναι ορισμένες στο αρχείο server/scripts/game.cs και οι αντίστοιχοι handlers περιέχονται στο αρχείο client/scripts/game.cs. Οι συναρτήσεις αυτές γράφτηκαν προκειμένου η εκτέλεσή τους να μπορεί να προκαλέσει συναισθήματα όπως: εχθουσιασμός, χαρά, λύπη, αγανάκτηση, ξάφνιασμα ευχάριστο ή δυσάρεστο.
Παρακάτω παρουσιάζονται στον πρώτο πίνακα οι συναρτήσεις, όπως ορίζοναται στην πλευρά του εξυπηρετητή και στο δεύτερο πίνακα οι αντίστοιχες από την πλευρά του πελάτη:

```
//Συνάρτηση που όταν καλείται αυξάνει το score του χρήστη κατά 1 ενώ παράλληλα
//ακούγεται και ήχος χειροκροτημάτων.
function serverIncreaseScore()
{
        for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ )
{
    %cl = ClientGroup.getObject( %clientIndex );
    commandToClient(%cl, 'PlayApplaus');
        }

}

//Συνάρτηση που όταν καλείται μειώνει το score του παίχτη κατά 1 ενώ παράλληλα
//ακούγεται και ήχος που έχει επιλεγεί για αποτυχία.
function serverDecreaseScore()
{
```

```
        for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ )
{
    %cl = ClientGroup.getObject( %clientIndex );
    commandToClient(%cl, 'Playminus1');
        }
}

//Συνάρτηση που όταν καλείται εμφανίζεται το μήνυμα της νίκης στον χρήστη
//συνοδευόμενο από αντίστοιχο ήχο.
function serverYouWin()
{
        for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ )
{
    %cl = ClientGroup.getObject( %clientIndex );
    commandToClient(%cl, 'loadyouWinGui');
        }

}

//Συνάρτηση που όταν καλέιται ακούγεται στο χρήστη ήχος που μοιάζει με ουρλιαχτό
function serverSound()
{
        for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ )
{
    %cl = ClientGroup.getObject( %clientIndex );
    commandToClient(%cl, 'PlayDeath');
        }

}

//Συνάρτηση που όταν εκτελείται ακούγεται ήχος κεραυνού
function serverSound2()
{
        for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ )
{
    %cl = ClientGroup.getObject( %clientIndex );
    commandToClient(%cl, 'PlayThunder');
        }

}

//Συνάρτηση που όταν καλείται ο παίχτης χάνει και εμφανίζεται κατάλληλο μήνυμα στην
//οθόνη του που συνοδεύεται από τον αντίστοιχο ήχο.
function serverGameOver()
{
```

```
        for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ )
{
    %cl = ClientGroup.getObject( %clientIndex );
    commandToClient(%cl, 'loadgameOverGui');
        }

}

//Συνάρτηση που όταν καλείται ακούγεται κορνάρισμα και εμφανίζεται μία εικόνα που
//προτρέπει το χρήστη να είναι πιο προσεχτικός ενώ το παιχνίδι επανεκκινεί.
function serverHorn()
{
        for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ )
{
    %cl = ClientGroup.getObject( %clientIndex );
    commandToClient(%cl, 'loadOpsGui');
        }

}
```

Συναρτήσεις ορισμένες στην πλευρά του εξυπηρετητή.

```
//Συνάρτηση handler για αύξηση του score κατά 1 και αναπαραγωγή ήχου.
function clientCmdPlayApplaus()
{
        alxPlay(Applaus);
        PlayGui.score++;
        PlayGui.updateScoreCounter();

}

//Συνάρτηση handler για μείωση του score κατά 1 και αναπαραγωγή ήχου.
function clientCmdPlayminus1()
{
        alxPlay(minus1);
        PlayGui.score--;
        PlayGui.updateScoreCounter();
}

//Συνάρτηση handler που φορτώνει στην οθόνη του χρήστη το γραφικό περιβάλλον της
//νίκης που έχει περιγραφεί σε προηγούμενο κεφάλαιο.
function clientCmdloadyouWinGui()
{
        loadyouWinGui();
}
```

```
//Συνάρτηση handler που αναπαράγει τον ήχο του ουρλιαχτού.
function clientCmdPlayDeath()
{
	alxPlay(Death);
}

//Συνάρτηση handler που αναπαράγει τον ήχο του κεραυνού.
function clientCmdPlayThunder()
{
	alxPlay(Thunder);
}

//Συνάρτηση handler που φορτώνει στην οθόνη του χρήστη το γραφικό περιβάλλον της
//ήττας που έχει περιγραφεί σε προηγούμενο κεφάλαιο.
function clientCmdloadgameOverGui()
{
	loadgameOverGui();
}

//Συνάρτηση handler που φορτώνει στην οθόνη του χρήστη το γραφικό περιβάλλον της
//προειδοποίησης για πιο προσεχτική οδήγηση με τη συνοδεία ήχου κορναρίσματος.
function clientCmdloadOpsGui()
{
	loadOpsGui();
}
```

Συναρτήσεις ορισμένες στην πλευρά του πελάτη

Η λειτουργία αυτών των συναρτήσεων είναι η ακόλουθη:

- **serverIncreaseScore():** Με κλήση αυτής της συνάρτησης από τον αφοσιωμένο εξυπηρετητή και ενεργοποίηση του αντίστοιχου 'handler' από τη μεριά του πελάτη, αυξάνεται το score κατά μία μονάδα στο χρήστη, ενέργεια η οποία συνοδεύεται από ήχο επευφημιών. Σκοπός μας είναι να προκαλέσουμε στο χρήστη ξάφνιασμα και χαρά.
- **serverDecreaseScore():** Με κλήση αυτής της συνάρτησης από τον αφοσιωμένο εξυπηρετητή και ενεργοποίηση του αντίστοιχου 'handler' από τη μεριά του πελάτη, μειώνεται το score κατά μία μονάδα στο χρήστη, ενέργεια η οποία συνοδεύεται από ήχο αποτυχίας. Σκοπός μας είναι να προκαλέσουμε στο χρήστη το συναίσθημα της λύπης ή του θυμού.
- **serverYouWin():** Η συγκεκριμένη συνάρτηση δίνει εντολή για να φορτωθεί το γραφικό περιβάλλον της νίκης στην οθόνη του χρήστη ενώ παράλληλα ακούγεται ήχος που υποδηλώνει επιτυχία. Σε αυτή την περίπτωση ελπίζουμε ο χρήστης να νιώσει χαρά.
- **serverSound():** Όταν εκτελείται η συγκεκριμένη συνάρτηση ακούγεται στον υπολογιστή του χρήστη ένας ήχος που ομοιάζει με ουρλιαχτό και σκοπό έχει να προκαλέσει ξάφνιασμα και πιθανόν τρόμο στον παίχτη.

- **serverSound2():** Σε αυτή την περίπτωση έχουμε ακριβώς την ίδια λειτουργία με την προηγούμενη συνάρτηση με μόνη διαφορά ότι ο ήχος που αναπαράγεται είναι ο ήχος ενός κεραυνού. Τα αισθήματα που αναμένουμε να δημιουργηθούν στο χρήστη είναι τα ίδια με τα προαναφερθέντα.
- **serverGameOver():** Η συγκεκριμένη συνάρτηση δίνει εντολή για να φορτωθεί το γραφικό περιβάλλον της ήττας στην οθόνη του χρήστη ενώ παράλληλα ακούγεται ήχος γέλιου που υποδηλώνει κοροϊδία. Σε αυτή την περίπτωση ελπίζουμε ο χρήστης να νιώσει θυμό.
- **serverHorn():** Η χρήση αυτής της συνάρτησης προκαλεί την αναπαραγωγή ήχου κόρνας αυτοκινήτου η οποία συνοδεύεται από γραφικό περιβάλλον που προειδοποιεί το χρήστη να προσέξει ενώ το παιχνίδι επανεκκινεί. Σκοπός είναι ο χρήστης να απορήσει και να εκπλεγεί. Η εικόνα αυτού του γραφικού περιβάλλοντος φαίνεται στη συνέχεια:

Εικόνα 5.8: Γραφικό περιβάλλον που φορτώνεται μετά από κλήση της serverHorn()

Παρακάτω παραθέτουμε κάποιες ενδεικτικές εικόνες που λήφθησαν σε μία πρακτική δοκιμή της εφαρμογής όπου ο χρήστης παίζει το παιχνίδι που δημιουργήσαμε και αντιμετωπίζει τα γεγονότα που προκαλούνται από κάποιον εξυπηρετητή:

Εικόνα 5.9: Αντίδραση χρήστη στη συνάρτηση ServerSound2()

Εικόνα 5.10: Αντίδραση χρήστη στη συνάρτηση serverIncreaseScore()

Στη συνέχεια θα αναφερθούμε στους τρόπους με τους οποίους μπορεί αυτή η εφαρμογή που αναπτύξαμε να αξιοποιηθεί, στα πλεονεκτήματα που παρουσιάζει σε σχέση με τις ήδη υπάρχουσες μεθόδους καθώς και στις μελλοντικές εξελίξεις που μπορούν να γίνουν.

6 ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΕΞΕΛΙΞΕΙΣ

6.1 Αξιολόγηση της μεθόδου

Όπως έχει ήδη διαπιστωθεί από την παραπάνω περιγραφή, η συγκεκριμένη μέθοδος που αναπύξαμε μπορεί να προκαλέσει αρκετά διαφορετικά συναισθήματα στο χρήστη με τρόπο αυθόρμητο, χωρίς όμως να παραβλέπουμε το γεγονός της διαφορετικότητας κάθε ατόμου και των διαφορετικών τρόπων αντίδρασης που μπορεί να προκύψουν και οι οποίοι ίσως να επηρεάζονται από την πρότερη ψυχολογική κατάσταση ή τη διαφορετική αντίληψη των ερεθισμάτων από το περιβάλλον.

Εάν ο ερευνητής έχει λάβει υπόψιν του αυτούς τους αστάθμητους παράγοντες που μπορούν να στερβλώσουν σε κάποιο βαθμό το αποτέλεσμα, θα μπορούσαμε να πούμε ότι η συγκεκριμένη μέθοδος είναι αρκετά ακριβής και αποτελεσματική τόσο λόγω της φυσικότητας του αποτελέσματος, όσο και λόγω της δυνατότητας που δίνει η κινηματογράφηση της εξέλιξης ολόκληρης της διαδικασίας να μελετήσουμε πλήρως την εξέλιξη της έκφρασης ανάλογα με το συναίσθημα που έχει βιώσει ο χρήστης και όχι αποσπασματικά frames.

Συνάμα, με τη βοήθεια κάποιων ηχητικών markers που συνοδεύουν κάθε απρόσμενο γεγονός και ενός αρχείου .log που περιλαμβάνει όλη τη δραστηριότητα που έλαβε χώρα κατά τη διάρκεια τη εφαρμογής, μπορούμε να εντοπίσουμε ακριβώς το χρόνο που συνέβη η αλλαγή στο περιβάλλον της εφαρμογής και τον τρόπο αντίδρασης του χρήστη τη στιγμή αυτή. Στη συνέχεια, λαμβάνοντας τα κατάλληλα στιγμιότυπα από αυτή τη μαγνητοσκόπιση μπορεί να γίνει η ανάλυση των χαρακτηριστικών με τις γνωστές μεθόδους MPEG-4 FBA και FACS (Facial Action Coding System).

Τέλος, ένα από τα βασικότερα θετικά στοιχεία της διαδικασίας που αναπτύξαμε είναι ότι η πρόκληση των συναισθημάτων γίνεται εντός ενός καθορισμένου πλαισίου που μας επιτρέπει να έχουμε πιο αξιόπιστα αποτελέσματα. Αυτό συμβαίνει γιατί ο σκοπός του κάθε χρήστη είναι η νίκη και δεν έχει όφελος να στρεβλώσει τις εκφράσεις του ή να μην εκφράσει τα πραγματικά του συναισθήματα όπως είναι πολύ πιθανόν να συμβεί σε μία διαδικασία χωρίς καθορισμένο περιεχόμενο.

6.2 Μελλοντικές εξελίξεις

Όπως έχει γίνει ήδη κατανοητό, μέσω της συγκεκριμένης εργασίας περιγράφεται μία μέθοδος πρόκλησης συναισθημάτων και αντιστοίχησής τους σε εκφράσεις του προσώπου, η οποία παρουσιάζει αρκετά πλεονεκτήματα σε σχέση με άλλες προϋπάρχουσες μεθόδους. Το επόμενο στάδιο στο οποίο θα πρέπει να εστιάσουν οι μελλοντικές εργασίες είναι η βελτίωση της διαδικασίας συλλογής δεδομένων ώστε να μπορεί να αντιστοιχηθεί το κάθε συναίσθημα, όχι απλά σε μία έκφραση του προσώπου αλλά και σε άλλα βιολογικά σήματα που μεταβάλλονται σύνφωνα με τη συναισθηματική κατάσταση. Θα ήταν χρήσιμο δηλαδή, να μπορούμε να συλλέγουμε τόσο παρεισφρητικά όσο και μη παρεισφρητικά στοιχεία, ώστε να μπορούμε να εξάγουμε ακριβή συμπεράσματα για τη συναισθηματική κατάσταση μέσα από πολλά κανάλια συλλογής δεδομένων.

Γι’ αυτό το λόγο θα πρέπει αρχικά να δημιουργηθούν πιο λειτουργικοί και εύχρηστοι αισθητήρες ώστε η χρήση τους να μη δυασχεραίνει την πειραματική διαδικασία. Ήδη έχουν γίνει βήματα βελτίωσης προς αυτή την κατεύθυνση καθώς οι αρχικοί ιατρικοί αισθητήρες που τοποθετούνταν στο χέρι, έχουν πλέον ενσωματωθεί

μέσα σε κοσμήματα, παπούτσια, ρούχα, γυαλιά ακόμα και σε ποντίκι υπολογιστή. Συνεπώς, όσο πιο εύχρηστοι, πρακτικοί και προσιτοί γίνουν οι αισθητήρες που είναι υπεύθυνοι για τη συλλογή των βιοσημάτων, τόσο πιο ακριβή, σφαιρικά και αξιόπιστα αποτελέσματα θα έχουμε. Βασικό λόγο φυσικά σε αυτή τη διαδικασία κατέχει η ανάλυση και η επεξεργασία των σημάτων από την οποία πρέπει να εξαχθούν οι χρήσιμες πληροφορίες και να αντιστοιχηθούν σε συγκεκριμένα εξωτερικά γεγονότα. Τέλος, η ανάπτυξη ενός έμπειρου συστήματος στο οποίο θα ενσωματωθεί όλη η γνώση που θα προκύψει από αυτά τα πειράματα, θα πραγματοποιήσει το τελικό βήμα για τη 'συναισθηματική εκπαίδευση' των υπολογιστικών συστημάτων.

ПАРАРТНМА

Στη συνέχεια παρατίθεται ο κώδικας από τα σημαντκότερα αρχεία της εφαρμογής τα οποία για διευκόλυνση της αναγνωσιμότητας του κειμένου δεν παρουσιάστηκαν ολοκληρωμένα στα παραπάνω κεφάλαια:

main.cs:

```
//-----------------------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//-----------------------------------------------------------------------------

// Load up common script base
loadDir("common");

// Defaults console values
exec("starter.racing/initialize.cs");
exec("./client/defaults.cs");
exec("./server/defaults.cs");
exec("./server/scripts/narki.cs");
exec("./server/scripts/healthkit.cs");


// Preferences (overide defaults)
exec("./client/prefs.cs");
exec("./server/prefs.cs");


// Package overrides to initialize the mod.
package FpsStarterKit {

function displayHelp() {
  Parent::displayHelp();
  error(
    "FPS Mod options:\n"@
    "  -dedicated            Start as dedicated server\n"@
    "  -connect <address>     For non-dedicated: Connect to a game at <address>\n" @
    "  -mission <filename>    For dedicated: Load the mission\n"
  );
}

function parseArgs()
{
  Parent::parseArgs();

  // Arguments, which override everything else.
  for (%i = 1; %i < $Game::argc ; %i++)
```

```
  {
    %arg = $Game::argv[%i];
    %nextArg = $Game::argv[%i+1];
    %hasNextArg = $Game::argc - %i > 1;

    switch$ (%arg)
    {
      //--------------------
      case "-dedicated":
        $Server::Dedicated = true;
        enableWinConsole(true);
        $argUsed[%i]++;

      //--------------------
      case "-mission":
        $argUsed[%i]++;
        if (%hasNextArg) {
          $missionArg = %nextArg;
          $argUsed[%i+1]++;
          %i++;
        }
        else
          error("Error: Missing Command Line argument. Usage: -mission <filename>");

      //--------------------
      case "-connect":
        $argUsed[%i]++;
        if (%hasNextArg) {
          $JoinGameAddress = %nextArg;
          $argUsed[%i+1]++;
          %i++;
        }
        else
          error("Error: Missing Command Line argument. Usage: -connect
<ip_address>");
    }
  }
}

function onStart()
{
  Parent::onStart();
  echo("\n--------- Initializing MOD: FPS Starter Kit ---------");

  // Load the scripts that start it all...
  exec("./client/init.cs");
```

```
   exec("./client/scripts/game.cs");
   exec("./server/init.cs");
   exec("client/data/init.cs");

   // Server gets loaded for all sessions, since clients
   // can host in-game servers.
   initServer();

   // Start up in either client, or dedicated server mode
   if ($Server::Dedicated)
      initDedicated();
   else
      initClient();
}

function onExit()
{
   echo("Exporting client prefs");
   export("$pref::*", "./client/prefs.cs", False);

   echo("Exporting client config");
   if (isObject(moveMap))
      moveMap.save("./client/config.cs", false);

   echo("Exporting server prefs");
   export("$Pref::Server::*", "./server/prefs.cs", False);
   BanList::Export("./server/banlist.cs");

   Parent::onExit();
}

}; // Client package
activatePackage(FpsStarterKit);
```

server/scripts/narki.cs:

```
//-----------------------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//-----------------------------------------------------------------------------


datablock ParticleData(narkiExplosionSparks)
{
   textureName          = "~/data/shapes/particles/spark";
   dragCoefficient      = 1;
```

```
   gravityCoefficient   = 0.0;
   inheritedVelFactor   = 0.2;
   constantAcceleration = 0.0;
   lifetimeMS           = 500;
   lifetimeVarianceMS   = 350;

   colors[0]     = "0.60 0.40 0.30 1.0";
   colors[1]     = "0.60 0.40 0.30 1.0";
   colors[2]     = "1.0 0.40 0.30 0.0";

   sizes[0]      = 0.25;
   sizes[1]      = 0.15;
   sizes[2]      = 0.15;

   times[0]      = 0.0;
   times[1]      = 0.5;
   times[2]      = 1.0;
};

datablock ParticleEmitterData(narkiExplosionSparkEmitter)
{
   ejectionPeriodMS = 3;
   periodVarianceMS = 0;
   ejectionVelocity = 5;
   velocityVariance = 1;
   ejectionOffset   = 0.0;
   thetaMin         = 0;
   thetaMax         = 180;
   phiReferenceVel  = 0;
   phiVariance      = 360;
   overrideAdvances = false;
   orientParticles  = true;
   lifetimeMS       = 100;
   particles = "narkiExplosionSparks";
};

datablock ParticleData(narkiExplosionFire)
{
   textureName          = "~/data/shapes/particles/fire";
   dragCoeffiecient     = 100.0;
   gravityCoefficient   = 0;
   inheritedVelFactor   = 0.25;
   constantAcceleration = 0.1;
   lifetimeMS           = 1200;
   lifetimeVarianceMS   = 300;
   useInvAlpha =  false;
```

```
   spinRandomMin = -80.0;
   spinRandomMax =  80.0;

   colors[0]     = "0.8 0.4 0 0.8";
   colors[1]     = "0.2 0.0 0 0.8";
   colors[2]     = "0.0 0.0 0.0 0.0";

   sizes[0]      = 1.5;
   sizes[1]      = 0.9;
   sizes[2]      = 0.5;

   times[0]      = 0.0;
   times[1]      = 0.5;
   times[2]      = 1.0;
};

datablock ParticleEmitterData(narkiExplosionFireEmitter)
{
   ejectionPeriodMS = 10;
   periodVarianceMS = 0;
   ejectionVelocity = 0.8;
   velocityVariance = 0.5;
   thetaMin         = 0.0;
   thetaMax         = 180.0;
   lifetimeMS       = 250;
   particles = "narkiExplosionFire";
};


datablock ParticleData(narkiExplosionSmoke)
{
   textureName          = "~/data/shapes/particles/smoke";
   dragCoeffiecient     = 100.0;
   gravityCoefficient   = 0;
   inheritedVelFactor   = 0.25;
   constantAcceleration = -0.30;
   lifetimeMS           = 1200;
   lifetimeVarianceMS   = 300;
   useInvAlpha =  true;
   spinRandomMin = -80.0;
   spinRandomMax =  80.0;

   colors[0]     = "0.56 0.36 0.26 1.0";
   colors[1]     = "0.2 0.2 0.2 1.0";
   colors[2]     = "0.0 0.0 0.0 0.0";
```

```
   sizes[0]      = 4.0;
   sizes[1]      = 2.5;
   sizes[2]      = 1.0;

   times[0]      = 0.0;
   times[1]      = 0.5;
   times[2]      = 1.0;
};
datablock ParticleEmitterData(narkiExplosionSmokeEmitter)
{
   ejectionPeriodMS = 10;
   periodVarianceMS = 0;
   ejectionVelocity = 4;
   velocityVariance = 0.5;
   thetaMin         = 0.0;
   thetaMax         = 180.0;
   lifetimeMS       = 250;
   particles = "narkiExplosionSmoke";
};

datablock ExplosionData(narkiSubExplosion1)
{
   offset = 0;
   emitter[0] = narkiExplosionSmokeEmitter;
   emitter[1] = narkiExplosionSparkEmitter;
};

datablock ExplosionData(narkiSubExplosion2)
{
   offset = 1.0;
   emitter[0] = narkiExplosionSmokeEmitter;
   emitter[1] = narkiExplosionSparkEmitter;
};



datablock ExplosionData(narkiExplosion)
{

   // Volume particles
   particleEmitter = narkiExplosionFireEmitter;
   particleDensity = 75;
   particleRadius = 2;
   soundProfile =AudioExplosion;

   // Point emission
```

```
   emitter[0] = narkiExplosionSmokeEmitter;
   emitter[1] = narkiExplosionSparkEmitter;

   // Sub explosion objects
   subExplosion[0] = narkiSubExplosion1;
   subExplosion[1] = narkiSubExplosion2;

   // Camera Shaking
   shakeCamera = true;
   camShakeFreq = "10.0 11.0 10.0";
   camShakeAmp = "1.0 1.0 1.0";
   camShakeDuration = 0.5;
   camShakeRadius = 10.0;

      // Impulse
   impulseRadius = 10;
   impulseForce = 15;

   // Dynamic light
   lightStartRadius = 6;
   lightEndRadius = 3;
   lightStartColor = "0.5 0.5 0";
   lightEndColor = "0 0 0";
};


  //orismos tis narkis
datablock StaticShapeData (narki)
{
        shapeFile = "~/data/shapes/buggy/simpleshape.dts";
        category = "Items";
        mass=1;
        friction=1;
        emap = true;
        damageAmount =20;
        explosion = narkiExplosion;
};
```

server/scripts/healthkit.cs:

```
//--------------------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//--------------------------------------------------------------------------
```

```
//orismos tou healthkit
datablock StaticShapeData(healthkit)

{
        shapeFile = "~/data/shapes/buggy/firstaid.dts";
        category = "Items";
        mass=1;
        friction=1;
        emap = true;
};

//Sunartisi sugkrousis oximatos me helathkit
function healthkit::onCollision(%this, %obj, %col)
{
  %col_className = %col.getClassname();
  %col_dblock_className =%col.getClassName();
  %colName = %col.getDataBlock().getName;
  if (%col.getClassName() $= "WheeledVehicle")
        {
    alxPlay(bonus);
    %client=%col.client;
        %client.score++;
        commandToClient(%client,'SetScoreCounter', %client.score);
        commandToClient(%client, 'PlayCollision2');
        %obj.delete();
                if (%client.score ==10)
                {
                        loadyouWinGui();
                         for( %clientIndex = 0; %clientIndex < ClientGroup.getCount();
%clientIndex++ ) {
           %cl = ClientGroup.getObject( %clientIndex );
           commandToClient(%cl, 'loadyouWinGui');
        }
                }
        }
}
```

client/scripts/audioProfiles.cs:

```
//--------------------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//--------------------------------------------------------------------------

// Channel assignments
```

```
$GuiAudioType     = 1;
$SimAudioType     = 2;
$MessageAudioType = 3;

new AudioDescription(AudioGui)
{
  volume   = 1.0;
  isLooping= false;
  is3D     = false;
  type     = $GuiAudioType;
};

new AudioDescription(AudioMessage)
{
  volume   = 1.0;
  isLooping= false;
  is3D     = false;
  type     = $MessageAudioType;
};

new AudioProfile(AudioButtonOver)
{
  filename = "client/data/sound/buttonOver.wav";
  description = "AudioGui";
        preload = true;
};

new AudioProfile(AudioCountBeep)
{
        filename = "~/data/sound/beep1.wav";
        description = "AudioGui";
        preload = true;
};

new AudioProfile(AudioGoBeep)
{
        filename = "~/data/sound/beep2.wav";
        description = "AudioGui";
        preload = true;
};

new AudioProfile(AudioExplosion)
{
        filename = "~/data/sound/Explosion.wav";
        description = "AudioGui";
        preload = true;
```

```
};

new AudioProfile(victory)
{
	filename = "~/data/sound/logo.wav";
	description = "AudioGui";
	preload = true;
};

new AudioProfile(death)
{
	filename = "~/data/sound/Death.wav";
	description = "AudioGui";
	preload = true;
};

new AudioProfile(thunder)
{
	filename = "~/data/sound/thunder.wav";
	description = "AudioGui";
	preload = true;
};

new AudioProfile(gameOver)
{
	filename = "~/data/sound/gameOver.wav";
	description = "AudioGui";
	preload = true;
};

new AudioProfile(Applaus)
{
	filename = "~/data/sound/Applaus.wav";
	description = "AudioGui";
	preload = true;
};

new AudioProfile(bonus)
{
	filename = "~/data/sound/healthkit.wav";
	description = "AudioGui";
	preload = true;
};

new AudioProfile(minus1)
{
```

```
	filename = "~/data/sound/decScore.wav";
	description = "AudioGui";
	preload = true;
};

new AudioProfile(DecHealth)
{
	filename = "~/data/sound/DecHealth.wav";
	description = "AudioGui";
	preload = true;
};
```

server/scripts/game.cs:

```
//-----------------------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//-----------------------------------------------------------------------------

// Pause while looking over the end game screen (in secs)
$Game::EndGamePause = 10;

// Pause until race starts.
$Game::WaitTime = 5000;

// The amount of laps to race through.
$Game::Laps = 2;

//-----------------------------------------------------------------------------
//  Functions that implement game-play
//-----------------------------------------------------------------------------

function onServerCreated()
{
   // Server::GameType is sent to the master server.
   // This variable should uniquely identify your game and/or mod.
   $Server::GameType = "Racing Starter Kit";

   // Server::MissionType sent to the master server.  Clients can
   // filter servers based on mission type.
   $Server::MissionType = "Racing";

   // GameStartTime is the sim time the game started. Used to calculated
   // game elapsed time.
   $Game::StartTime = 0;
```

```
   // Load up all datablocks, objects etc.  This function is called when
   // a server is constructed.
   exec("./audioProfiles.cs");
   exec("./camera.cs");
   exec("./markers.cs");
   exec("./triggers.cs");
   exec("./shapeBase.cs");
   exec("./staticShape.cs");
   exec("./narki.cs");
   exec("./healthkit.cs");
   exec("./radiusDamage.cs");
   exec("./car.cs");
   exec("./checkpoint.cs");

   // Keep track of when the game started
   $Game::StartTime = $Sim::Time;
}

function onServerDestroyed()
{
   // This function is called as part of a server shutdown.
}

//-----------------------------------------------------------------------------

function onMissionLoaded()
{
   // Called by loadMission() once the mission is finished loading.
   // Nothing special for now, just start up the game play.
   startGame();
}

function onMissionEnded()
{
   // Called by endMission(), right before the mission is destroyed

   // Normally the game should be ended first before the next
   // mission is loaded, this is here in case loadMission has been
   // called directly.  The mission will be ended if the server
   // is destroyed, so we only need to cleanup here.
   cancel($Game::Schedule);
   $Game::Running = false;
   $Game::Cycling = false;
}
```

```
//-----------------------------------------------------------------------------

function startGame()
{
  if ($Game::Running) {
    error("startGame: End the game first!");
    return;
  }

  }

function countThree()
{
  // Tell the client to count.
  for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ ) {
    %cl = ClientGroup.getObject( %clientIndex );
    commandToClient(%cl, 'PlayCounter', 3);
  }
          schedule(1000, 0, "countTwo");
}

function countTwo()
{
  // Tell the client to count.
 for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ ) {
    %cl = ClientGroup.getObject( %clientIndex );
    commandToClient(%cl, 'PlayCounter', 2);
  }
        schedule(1000, 0, "countOne");
}

function countOne()
{
  // Tell the client to count.
  for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ ) {
    %cl = ClientGroup.getObject( %clientIndex );
    commandToClient(%cl, 'PlayCounter', 1);
  }
        schedule(1000, 0, "startRace");
}

function startRace()
{
  for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ ) {
    %cl = ClientGroup.getObject( %clientIndex );
    commandToClient(%cl, 'PlayStart');
```

```
   }

 // Other client specific setup..
      %cl.lap = 0;
      %cl.nextCheck = 1;
      %cl.setControlObject(%cl.car);
      %cl.camera.setFlyMode();
      $Game::Running = true;

}

function endGame()
{
   if (!$Game::Running)  {
      error("endGame: No game running!");
      return;
   }

   // Stop any game timers
   cancel($Game::Schedule);

   // Inform the client the game is over
   for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ ) {
      %cl = ClientGroup.getObject( %clientIndex );
      commandToClient(%cl, 'GameEnd');
   }

   // Delete all the temporary mission objects
   resetMission();
   $Game::Running = false;
}

function onGameDurationEnd()
{
   // This "redirect" is here so that we can abort the game cycle if
   // the $Game::Duration variable has been cleared, without having
   // to have a function to cancel the schedule.
   if ($Game::Duration && !isObject(EditorGui))
      cycleGame();
}

//------------------------------------------------------------------------

function cycleGame()
{
   // This is setup as a schedule so that this function can be called
```

```
   // directly from object callbacks.  Object callbacks have to be
   // carefull about invoking server functions that could cause
   // their object to be deleted.
   if (!$Game::Cycling) {
      $Game::Cycling = true;
      $Game::Schedule = schedule(0, 0, "onCycleExec");
   }
}

function onCycleExec()
{
   // End the current game and start another one, we'll pause for a little
   // so the end game victory screen can be examined by the clients.
   endGame();
   $Game::Schedule = schedule($Game::EndGamePause * 1000, 0, "onCyclePauseEnd");
}

function onCyclePauseEnd()
{
   $Game::Cycling = false;

   // Just cycle through the missions for now.
   %search = $Server::MissionFileSpec;
   for (%file = findFirstFile(%search); %file !$= ""; %file = findNextFile(%search)) {
      if (%file $= $Server::MissionFile) {
         // Get the next one, back to the first if there
         // is no next.
         %file = findNextFile(%search);
         if (%file $= "")
           %file = findFirstFile(%search);
         break;
      }
   }
   loadMission(%file);
}


//--------------------------------------------------------------------------
// GameConnection Methods
// These methods are extensions to the GameConnection class. Extending
// GameConnection make is easier to deal with some of this functionality,
// but these could also be implemented as stand-alone functions.
//--------------------------------------------------------------------------

function GameConnection::onClientEnterGame(%this)
{
```

```
  commandToClient(%this, 'SyncClock', $Sim::Time - $Game::StartTime);
  commandToClient(%this, 'SetMaxLaps', $Game::Laps);

  for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ ) {
    %cl = ClientGroup.getObject( %clientIndex );

        schedule(1000, 0, "countThree");

        }


  //Gia na ksekinaei me tin camera ektos oximatos kai o client -mpainei entos oximatos
//patwntas to tab
  %this.setFirstPerson(false);

  // Create a new camera object.
  %this.camera = new Camera() {
    dataBlock = Observer;
  };
  MissionCleanup.add( %this.camera );
  %this.camera.scopeToClient(%this);

  //Gia na ksekinaei me tin camera ektos oximatos - mpainei sto oxima patwntas to tab
  ServerConnection.setFirstPerson(false);

  // Client controls the camera by default.
  %this.setControlObject(%this.camera);

            // Create a car object.
            %this.spawnCar();

     //Otan ksekinaei o pelatis exei elegxo tou autokinitou kai oxi tis cameras (de
//xreiazetai to tab)
            %this.setControlObject(%this.car);
            %this.camera.setOrbitMode(%this.car, %this.car.getTransform(), 0.5, 4.5,
4.5);

}

function GameConnection::onClientLeaveGame(%this)
{
  if (isObject(%this.camera))
    %this.camera.delete();
  if (isObject(%this.car))
    %this.car.delete();
}
```

```
//--------------------------------------------------------------------------

function GameConnection::spawnCar(%this)
{
  // Combination create player and drop him somewhere
  %spawnPoint = pickSpawnPoint();
  %this.createCar(%spawnPoint);
}

//--------------------------------------------------------------------------

function GameConnection::createCar(%this, %spawnPoint)
{
  if (%this.car > 0)  {
    // The client should not have a player currently
    // assigned.  Assigning a new one could result in
    // a player ghost.
    error( "Attempting to create an angus ghost!" );
          %this.setHealthAmountHud(100);
  }

  // Create the player object
  %car = new WheeledVehicle() {
    dataBlock = DefaultCar;
    client = %this;
  };
  MissionCleanup.add(%car);

  // Player setup...
  %car.setTransform(%spawnPoint);
  %car.setShapeName(%this.name);

  // Update the camera to start with the player
  %this.camera.setTransform(%car.getEyeTransform());

  // Give the client control of the player
  %this.car = %car;
}


function narki::onCollision( %this, %obj, %col ){
  if( %col.getType() & $TypeMasks::ShapeBaseObjectType )    {
    %col.damage( %obj, %obj.getPosition(), 50, "Mine" );
    %obj.setDamageState(Destroyed);
    %obj.schedule(200, "delete");
```

```
            %client=%col.client;
            %client.Health =%client.Health-20;
    commandToClient(%client, 'PlayCollision', %obj);
          commandToClient(%client,'SetHealthCounter', %client.Health);
          if (%client.Health ==-100)
                {
                      loadgameOverGui();
                      for( %clientIndex = 0; %clientIndex < ClientGroup.getCount();
%clientIndex++ ) {
     %cl = ClientGroup.getObject( %clientIndex );
     commandToClient(%cl, 'loadgameOverGui');
          }
                }
          }
}


function narki::onDestroyed(%data, %obj, %pos, %mod)
{
  %pos = %obj.getPosition();
  radiusDamage( %obj, %pos, 4.0 ,%data.radiusDamage, "MineRadius",
%data.areaImpulse );
}

//--------------------------------------------------------------------------
// Support functions
//--------------------------------------------------------------------------

function pickSpawnPoint()
{
  %groupName = "MissionGroup/PlayerDropPoints";
  %group = nameToID(%groupName);

  if (%group != -1) {
    %count = %group.getCount();
    if (%count != 0) {
      %index = getRandom(%count-1);
      %spawn = %group.getObject(%index);
      return %spawn.getTransform();
    }
    else
      error("No spawn points found in " @ %groupName);
  }
  else
    error("Missing spawn points group " @ %groupName);
```

```
   // Could be no spawn points, in which case we'll stick the
   // player at the center of the world.
   return "0 0 300 1 0 0 0";
}


function serverincreaselap()
{
          for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ )
{
     %cl = ClientGroup.getObject( %clientIndex );
     commandToClient(%cl, 'IncreaseLapCounter');
          }
          PlayGui.lap++;

}

function serverHorn()
{
          for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ )
{
     %cl = ClientGroup.getObject( %clientIndex );
     commandToClient(%cl, 'loadOpsGui');
          }

}

//Sunartisi pou otan kaleitai apo to server auksanetai to score
//kata 1.
function serverIncreaseScore()
{
          for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ )
{
     %cl = ClientGroup.getObject( %clientIndex );
     commandToClient(%cl, 'PlayApplaus');
          }

}

//Sunartisi pou otan kaleitai apo to server meiwnetai to score
//kata 1.
function serverDecreaseScore()
{
          for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ )
{
     %cl = ClientGroup.getObject( %clientIndex );
```

```
    commandToClient(%cl, 'Playminus1');
        }
}

//Sunartisi pou otan kaleitai apo to server emfanizetai to
//minima nikis You Win!!
function serverYouWin()
{
        for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ )
{
    %cl = ClientGroup.getObject( %clientIndex );
    commandToClient(%cl, 'loadyouWinGui');
        }

}

function serverSound()
{
        for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ )
{
    %cl = ClientGroup.getObject( %clientIndex );
    commandToClient(%cl, 'PlayDeath');
        }

}

function serverSound2()
{
        for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ )
{
    %cl = ClientGroup.getObject( %clientIndex );
    commandToClient(%cl, 'PlayThunder');
        }

}

function serverGameOver()
{
        for( %clientIndex = 0; %clientIndex < ClientGroup.getCount(); %clientIndex++ )
{
    %cl = ClientGroup.getObject( %clientIndex );
    commandToClient(%cl, 'loadgameOverGui');
        }

}
```

client/scripts/game.cs:

```
//-----------------------------------------------------------------------------
// Torque Game Engine
// Copyright (C) GarageGames.com, Inc.
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Game start / end events sent from the server
//-----------------------------------------------------------------------------

function clientCmdPlayCounter(%count)
{
	alxPlay(AudioCountBeep);
	if(%count == 3)
		counter.setVisible(true);

	counter.setBitmap("starter.racing/client/ui/" @ %count @ ".png");


}

function clientCmdSetScoreCounter(%score)
{
	ScoreCounter.setText("Score:" SPC %score);
}

function clientCmdSetHealthCounter(%Health)
{
	%Health=%Health+100;
	HealthCounter.setText("Health:" SPC %Health);
}


function clientCmdPlayCollision(%obj)
{

	 alxplay(AudioExplosion);
}


function clientCmdPlayCollision2(%obj)
{
	alxPlay(bonus);

}
```

```
function clientCmdPlayStart(%seq)
{
	alxPlay(AudioGoBeep);
	// Display the GO bitmap and play a sound
	counter.setBitmap("starter.racing/client/ui/go.png");
	// Remove the GO after a second.
	counter.schedule(1000, setVisible, false);
}


function clientCmdGameEnd(%seq)
{
  // Stop local activity... the game will be destroyed on the server
  alxStopAll();

  // Copy the current scores from the player list into the
  // end game gui (bit of a hack for now).
  EndGameGuiList.clear();
  for (%i = 0; %i < PlayerListGuiList.rowCount(); %i++) {
    %text = PlayerListGuiList.getRowText(%i);
    %id = PlayerListGuiList.getRowId(%i);
    EndGameGuiList.addRow(%id,%text);
  }
  EndGameGuiList.sortNumerical(1,false);

  // Display the end-game screen
  Canvas.setContent(EndGameGui);
}

function clientCmdIncreaseLapCounter()
{
	// Increase the lap.
	PlayGui.lap++;
	PlayGui.updateLapCounter();
}

function clientCmdPlayApplaus()
{
	// Increase the score.
	alxPlay(Applaus);
	PlayGui.score++;
	PlayGui.updateScoreCounter();

}
```

```
function clientCmdPlayminus1()
{
	// Decrease the score.
	alxPlay(minus1);
	PlayGui.score--;
	PlayGui.updateScoreCounter();
}

function clientCmdloadOpsGui()
{

	loadOpsGui();

}


function clientCmdloadyouWinGui()
{

	loadyouWinGui();
}

function clientCmdPlayDeath()
{

	alxPlay(Death);
}

function clientCmdPlayThunder()
{

	alxPlay(Thunder);
}

function clientCmdloadgameOverGui()
{

	loadgameOverGui();
}
```

data/missions/racing.mis:

```
//--- OBJECT WRITE BEGIN ---
new SimGroup(MissionGroup) {
   canSaveDynamicFields = "1";
```

```
new ScriptObject(MissionInfo) {
      desc0 = "This is a simple racing example mission.";
      name = "Racing Example";
      type = "racing";
};
new MissionArea(MissionArea) {
   canSaveDynamicFields = "1";
   Area = "-1024 -1024 2048 2048";
   flightCeiling = "300";
   flightCeilingRange = "20";
      locked = "true";
};
new SimGroup(environment) {
   canSaveDynamicFields = "1";

   new Sky(Sky) {
      canSaveDynamicFields = "1";
      position = "336 136 0";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      materialList = "~/data/skies/sky_sunset1.dml";
      cloudHeightPer[0] = "0.1";
      cloudHeightPer[1] = "0.1";
      cloudHeightPer[2] = "0.1";
      cloudSpeed1 = "0.0005";
      cloudSpeed2 = "0.0005";
      cloudSpeed3 = "0.0005";
      visibleDistance = "1000";
      fogDistance = "300";
      fogColor = "1 0.8 0.7 1";
      fogStorm1 = "0";
      fogStorm2 = "0";
      fogStorm3 = "0";
      fogVolume1 = "0 0 0";
      fogVolume2 = "0 0 0";
      fogVolume3 = "0 0 0";
      fogVolumeColor1 = "128 128 128 -2.22768e+038";
      fogVolumeColor2 = "128 128 128 0";
      fogVolumeColor3 = "128 128 128 -1.70699e+038";
      windVelocity = "1 1 0";
      windEffectPrecipitation = "1";
      SkySolidColor = "0.6 0.6 0.6 1";
      useSkyTextures = "1";
      renderBottomTexture = "0";
      noRenderBans = "0";
   };
```

```
new Sun() {
   canSaveDynamicFields = "1";
   azimuth = "50";
   elevation = "30";
   color = "0.6 0.5 0.45 1";
   ambient = "0.4 0.3 0.3 1";
   CastsShadows = "1";
      scale = "1 1 1";
      direction = "0.57735 0.57735 -0.57735";
      position = "0 0 0";
      rotation = "1 0 0 0";
};
new fxSunLight(sunflare1) {
   canSaveDynamicFields = "1";
   position = "98.5277 -289.053 188.13";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   Enable = "1";
   LocalFlareBitmap = "common/lighting/corona";
   RemoteFlareBitmap = "common/lighting/corona";
   SunAzimuth = "224";
   SunElevation = "10";
   LockToRealSun = "1";
   FlareTP = "1";
   Colour = "0.3 0.2 0 1";
   Brightness = "0.1";
   FlareSize = "2";
   FadeTime = "0.1";
   BlendMode = "0";
   AnimColour = "0";
   AnimBrightness = "1";
   AnimRotation = "1";
   AnimSize = "0";
   AnimAzimuth = "0";
   AnimElevation = "0";
   LerpColour = "1";
   LerpBrightness = "1";
   LerpRotation = "1";
   LerpSize = "1";
   LerpAzimuth = "1";
   LerpElevation = "1";
   LinkFlareSize = "0";
   SingleColourKeys = "1";
   MinColour = "0 0 0 1";
   MaxColour = "1 1 1 1";
   MinBrightness = "0.7";
```

```
    MaxBrightness = "1";
    MinRotation = "0";
    MaxRotation = "359";
    minSize = "1";
    MaxSize = "1";
    MinAzimuth = "0";
    MaxAzimuth = "359";
    MinElevation = "-30";
    MaxElevation = "210";
    RedKeys = "AZA";
    GreenKeys = "AZA";
    BlueKeys = "AZA";
    BrightnessKeys = "JAZJTAJ";
    RotationKeys = "ZA";
    SizeKeys = "ATAZA";
    AzimuthKeys = "AZ";
    ElevationKeys = "AZ";
    ColourTime = "5";
    BrightnessTime = "10";
    RotationTime = "40";
    SizeTime = "30";
    AzimuthTime = "5";
    ElevationTime = "5";
  };
  new fxSunLight(sunflare2) {
    canSaveDynamicFields = "1";
    position = "72.9234 -289.051 187.692";
    rotation = "1 0 0 0";
    scale = "1 1 1";
    Enable = "1";
    LocalFlareBitmap = "common/lighting/corona";
    RemoteFlareBitmap = "common/lighting/corona";
    SunAzimuth = "224";
    SunElevation = "10";
    LockToRealSun = "1";
    FlareTP = "1";
    Colour = "0.3 0.2 0.1 1";
    Brightness = "0.6";
    FlareSize = "0.6";
    FadeTime = "0.1";
    BlendMode = "0";
    AnimColour = "0";
    AnimBrightness = "0";
    AnimRotation = "1";
    AnimSize = "0";
    AnimAzimuth = "0";
```

```
      AnimElevation = "0";
      LerpColour = "1";
      LerpBrightness = "1";
      LerpRotation = "1";
      LerpSize = "1";
      LerpAzimuth = "1";
      LerpElevation = "1";
      LinkFlareSize = "0";
      SingleColourKeys = "1";
      MinColour = "0 0 0 1";
      MaxColour = "1 1 1 1";
      MinBrightness = "0.25";
      MaxBrightness = "0.5";
      MinRotation = "0";
      MaxRotation = "359";
      minSize = "3";
      MaxSize = "3";
      MinAzimuth = "0";
      MaxAzimuth = "359";
      MinElevation = "-30";
      MaxElevation = "210";
      RedKeys = "AZA";
      GreenKeys = "AZA";
      BlueKeys = "AZA";
      BrightnessKeys = "AZJTA";
      RotationKeys = "AZ";
      SizeKeys = "ATAZA";
      AzimuthKeys = "AZ";
      ElevationKeys = "AZ";
      ColourTime = "5";
      BrightnessTime = "6";
      RotationTime = "80";
      SizeTime = "5";
      AzimuthTime = "5";
      ElevationTime = "5";
   };
   new TerrainBlock(Terrain) {
      canSaveDynamicFields = "1";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      detailTexture = "~/data/terrains/details/detail1";
      terrainFile = "./racing.ter";
      squareSize = "8";
      bumpScale = "1";
      bumpOffset = "0.01";
      zeroBumpScale = "8";
```

```
      tile = "1";
         locked = "true";
         position = "-1024 -1024 0";
   };
   new fxFoliageReplicator(ShortGrass3) {
      canSaveDynamicFields = "1";
      position = "-204.949 986.166 199.214";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      UseDebugInfo = "0";
      DebugBoxHeight = "1";
      HideFoliage = "0";
      ShowPlacementArea = "0";
      PlacementAreaHeight = "25";
      PlacementColour = "0.4 0 0.8 1";
      Seed = "1376312589";
      FoliageFile = "~/data/shapes/plants/plant2";
      FoliageCount = "1000";
      FoliageRetries = "100";
      InnerRadiusX = "0";
      InnerRadiusY = "0";
      OuterRadiusX = "30";
      OuterRadiusY = "40";
      MinWidth = "0.5";
      MaxWidth = "3";
      MinHeight = "1";
      MaxHeight = "3";
      FixAspectRatio = "1";
      FixSizeToMax = "0";
      OffsetZ = "0";
      RandomFlip = "1";
      UseCulling = "1";
      CullResolution = "10";
      ViewDistance = "120";
      ViewClosest = "1";
      FadeInRegion = "40";
      FadeOutRegion = "1";
      AlphaCutoff = "0.1";
      GroundAlpha = "1";
      SwayOn = "1";
      SwaySync = "0";
      SwayMagSide = "0.01";
      SwayMagFront = "0.03";
      MinSwayTime = "5";
      MaxSwayTime = "10";
      LightOn = "1";
```

```
      LightSync = "0";
      MinLuminance = "0.7";
      MaxLuminance = "1";
      LightTime = "5";
      AllowOnTerrain = "1";
      AllowOnInteriors = "0";
      AllowOnStatics = "0";
      AllowOnWater = "0";
      AllowWaterSurface = "0";
      AllowedTerrainSlope = "90";
   };
   new fxFoliageReplicator(ShortGrass2) {
      canSaveDynamicFields = "1";
      position = "-178.909 470.781 156.266";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      UseDebugInfo = "0";
      DebugBoxHeight = "1";
      HideFoliage = "0";
      ShowPlacementArea = "0";
      PlacementAreaHeight = "25";
      PlacementColour = "0.4 0 0.8 1";
      Seed = "1376312589";
      FoliageFile = "~/data/shapes/plants/plant2";
      FoliageCount = "1000";
      FoliageRetries = "100";
      InnerRadiusX = "0";
      InnerRadiusY = "0";
      OuterRadiusX = "30";
      OuterRadiusY = "70";
      MinWidth = "0.5";
      MaxWidth = "3";
      MinHeight = "1";
      MaxHeight = "3";
      FixAspectRatio = "1";
      FixSizeToMax = "0";
      OffsetZ = "0";
      RandomFlip = "1";
      UseCulling = "1";
      CullResolution = "10";
      ViewDistance = "120";
      ViewClosest = "1";
      FadeInRegion = "40";
      FadeOutRegion = "1";
      AlphaCutoff = "0.1";
      GroundAlpha = "1";
```

```
   SwayOn = "1";
   SwaySync = "0";
   SwayMagSide = "0.01";
   SwayMagFront = "0.03";
   MinSwayTime = "5";
   MaxSwayTime = "10";
   LightOn = "1";
   LightSync = "0";
   MinLuminance = "0.7";
   MaxLuminance = "1";
   LightTime = "5";
   AllowOnTerrain = "1";
   AllowOnInteriors = "0";
   AllowOnStatics = "0";
   AllowOnWater = "0";
   AllowWaterSurface = "0";
   AllowedTerrainSlope = "90";
};
new fxFoliageReplicator(ShortGrass) {
   canSaveDynamicFields = "1";
   position = "-386.947 796.221 154.184";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   UseDebugInfo = "0";
   DebugBoxHeight = "1";
   HideFoliage = "0";
   ShowPlacementArea = "0";
   PlacementAreaHeight = "25";
   PlacementColour = "0.4 0 0.8 1";
   Seed = "1376312589";
   FoliageFile = "~/data/shapes/plants/plant2";
   FoliageCount = "1000";
   FoliageRetries = "100";
   InnerRadiusX = "0";
   InnerRadiusY = "0";
   OuterRadiusX = "30";
   OuterRadiusY = "40";
   MinWidth = "0.5";
   MaxWidth = "3";
   MinHeight = "1";
   MaxHeight = "3";
   FixAspectRatio = "1";
   FixSizeToMax = "0";
   OffsetZ = "0";
   RandomFlip = "1";
   UseCulling = "1";
```

```
      CullResolution = "10";
      ViewDistance = "120";
      ViewClosest = "1";
      FadeInRegion = "40";
      FadeOutRegion = "1";
      AlphaCutoff = "0.1";
      GroundAlpha = "1";
      SwayOn = "1";
      SwaySync = "0";
      SwayMagSide = "0.01";
      SwayMagFront = "0.03";
      MinSwayTime = "5";
      MaxSwayTime = "10";
      LightOn = "1";
      LightSync = "0";
      MinLuminance = "0.7";
      MaxLuminance = "1";
      LightTime = "5";
      AllowOnTerrain = "1";
      AllowOnInteriors = "0";
      AllowOnStatics = "0";
      AllowOnWater = "0";
      AllowWaterSurface = "0";
      AllowedTerrainSlope = "90";
   };
};
new SimGroup(PlayerDropPoints) {
   canSaveDynamicFields = "1";

   new SpawnSphere() {
      canSaveDynamicFields = "1";
      position = "-173.824 910.624 196.718";
      rotation = "0 0 -1 2.47371";
      scale = "1 1 1";
      dataBlock = "SpawnSphereMarker";
      Radius = "1";
      sphereWeight = "100";
      indoorWeight = "100";
      outdoorWeight = "100";
         locked = "false";
         lockCount = "0";
         homingCount = "0";
   };
   new SpawnSphere() {
      canSaveDynamicFields = "1";
      position = "-160.802 910.516 196.262";
```

```
      rotation = "0 0 -1 2.47371";
      scale = "1 1 1";
      dataBlock = "SpawnSphereMarker";
      Radius = "1";
      sphereWeight = "100";
      indoorWeight = "100";
      outdoorWeight = "100";
         locked = "false";
         lockCount = "0";
         homingCount = "0";
   };
   new SpawnSphere() {
      canSaveDynamicFields = "1";
      position = "-148.168 911.136 194.677";
      rotation = "0 0 -1 10.4951";
      scale = "1 1 1";
      dataBlock = "SpawnSphereMarker";
      Radius = "1";
      sphereWeight = "100";
      indoorWeight = "100";
      outdoorWeight = "100";
         locked = "false";
         lockCount = "0";
         homingCount = "0";
   };
};
new SimGroup(Buildings) {
   canSaveDynamicFields = "1";
};
new InteriorInstance() {
   canSaveDynamicFields = "1";
   position = "-131.795 917.09 192.98";
   rotation = "0 0 -1 8.02137";
   scale = "1 1 1";
   interiorFile = "~/data/interiors/racing/barrier1.dif";
   useGLLighting = "0";
   showTerrainInside = "0";
};
new InteriorInstance() {
   canSaveDynamicFields = "1";
   position = "-132.712 882.821 193.112";
   rotation = "0 0 1 9.74035";
   scale = "1 1 1";
   interiorFile = "~/data/interiors/racing/barrier1.dif";
   useGLLighting = "0";
   showTerrainInside = "0";
```

```
};
new InteriorInstance() {
   canSaveDynamicFields = "1";
   position = "-210.068 1114.36 158.395";
   rotation = "0 0 -1 112.873";
   scale = "1 1 1";
   interiorFile = "~/data/interiors/racing/barrier1.dif";
   useGLLighting = "0";
   showTerrainInside = "0";
};
new InteriorInstance() {
   canSaveDynamicFields = "1";
   position = "-334.644 848.543 154.841";
   rotation = "-0.152803 -0.142578 0.977918 95.2405";
   scale = "1 1 1";
   interiorFile = "~/data/interiors/racing/wedge.dif";
   useGLLighting = "0";
   showTerrainInside = "0";
};
new InteriorInstance() {
   canSaveDynamicFields = "1";
   position = "-213.138 568.133 147.487";
   rotation = "0.116233 -0.109843 -0.987129 86.6763";
   scale = "1 1 1";
   interiorFile = "~/data/interiors/racing/wedge.dif";
   useGLLighting = "0";
   showTerrainInside = "0";
};
new InteriorInstance() {
   canSaveDynamicFields = "1";
   position = "-357.272 497.436 164.997";
   rotation = "0 0 1 112.873";
   scale = "1 1 1";
   interiorFile = "~/data/interiors/racing/arrowsign.dif";
   useGLLighting = "0";
   showTerrainInside = "0";
};
new InteriorInstance() {
   canSaveDynamicFields = "1";
   position = "-220.044 476.387 155.583";
   rotation = "0 0 1 20.0538";
   scale = "1 1 1";
   interiorFile = "~/data/interiors/racing/arrowsign.dif";
   useGLLighting = "0";
   showTerrainInside = "0";
};
```

```
new InteriorInstance() {
   canSaveDynamicFields = "1";
   position = "-156.644 1104.9 170.703";
   rotation = "0 0 -1 90.527";
   scale = "1 1 1";
   interiorFile = "~/data/interiors/racing/arrowsign.dif";
   useGLLighting = "0";
   showTerrainInside = "0";
};
new InteriorInstance() {
   canSaveDynamicFields = "1";
   position = "-246.636 1089.27 162.629";
   rotation = "0 0 1 204.156";
   scale = "1 1 1";
   interiorFile = "~/data/interiors/racing/arrowsign.dif";
   useGLLighting = "0";
   showTerrainInside = "0";
};
new InteriorInstance() {
   canSaveDynamicFields = "1";
   position = "-221.443 830.139 202.262";
   rotation = "0 0 1 20.0538";
   scale = "1 1 1";
   interiorFile = "~/data/interiors/racing/arrowsign.dif";
   useGLLighting = "0";
   showTerrainInside = "0";
};
new InteriorInstance() {
   canSaveDynamicFields = "1";
   position = "-347.691 751.557 149.026";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   interiorFile = "~/data/interiors/racing/block1.dif";
   useGLLighting = "0";
   showTerrainInside = "0";
};
new InteriorInstance() {
   canSaveDynamicFields = "1";
   position = "-325.396 710.21 151.1";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   interiorFile = "~/data/interiors/racing/block1.dif";
   useGLLighting = "0";
   showTerrainInside = "0";
};
new InteriorInstance() {
```

```
   canSaveDynamicFields = "1";
   position = "-182.116 836.796 193.984";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   interiorFile = "~/data/interiors/racing/block1.dif";
   useGLLighting = "0";
   showTerrainInside = "0";
};
new InteriorInstance() {
   canSaveDynamicFields = "1";
   position = "-204.327 811.909 189.803";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   interiorFile = "~/data/interiors/racing/block1.dif";
   useGLLighting = "0";
   showTerrainInside = "0";
};
new InteriorInstance() {
   canSaveDynamicFields = "1";
   position = "-197.236 503.298 149.057";
   rotation = "0 0 1 6.87505";
   scale = "1 1 1";
   interiorFile = "~/data/interiors/racing/barrier1.dif";
   useGLLighting = "0";
   showTerrainInside = "0";
};
new InteriorInstance() {
   canSaveDynamicFields = "1";
   position = "-176.757 788.581 192.967";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   interiorFile = "~/data/interiors/racing/block1.dif";
   useGLLighting = "0";
   showTerrainInside = "0";
};
new InteriorInstance() {
   canSaveDynamicFields = "1";
   position = "-342.687 926.283 158.366";
   rotation = "0 0 1 189.832";
   scale = "1 1 1";
   interiorFile = "~/data/interiors/racing/arrowsign.dif";
   useGLLighting = "0";
   showTerrainInside = "0";
};
new Trigger(checkpoint4) {
   canSaveDynamicFields = "1";
```

```
    position = "-197.895 872.003 186.979";
    rotation = "1 0 0 0";
    scale = "60 5 20";
    dataBlock = "CheckPointTrigger";
    polyhedron = "0.0000000 0.0000000 0.0000000 1.0000000 0.0000000 0.0000000
0.0000000 -1.0000000 0.0000000 0.0000000 0.0000000 1.0000000";
      isLast = "1";
      checkpoint = "4";
  };
  new Trigger(checkpoint1) {
    canSaveDynamicFields = "1";
    position = "-286.543 1041.54 156.512";
    rotation = "0 0 1 26.356";
    scale = "60 5 20";
    dataBlock = "CheckPointTrigger";
    polyhedron = "0.0000000 0.0000000 0.0000000 1.0000000 0.0000000 0.0000000
0.0000000 -1.0000000 0.0000000 0.0000000 0.0000000 1.0000000";
      checkpoint = "1";
  };
  new Trigger(checkpoint2) {
    canSaveDynamicFields = "1";
    position = "-371.272 630.384 155.087";
    rotation = "1 0 0 0";
    scale = "60 5 20";
    dataBlock = "CheckPointTrigger";
    polyhedron = "0.0000000 0.0000000 0.0000000 1.0000000 0.0000000 0.0000000
0.0000000 -1.0000000 0.0000000 0.0000000 0.0000000 1.0000000";
      checkpoint = "2";
  };
  new Trigger(checkpoint3) {
    canSaveDynamicFields = "1";
    position = "-264.127 518.151 145.923";
    rotation = "0 0 1 89.3814";
    scale = "60 5 20";
    dataBlock = "CheckPointTrigger";
    polyhedron = "0.0000000 0.0000000 0.0000000 1.0000000 0.0000000 0.0000000
0.0000000 -1.0000000 0.0000000 0.0000000 0.0000000 1.0000000";
      checkpoint = "3";
  };
  new SpawnSphere() {
    canSaveDynamicFields = "1";
    position = "-142.323 898.383 192.945";
    rotation = "0 0 -1 10.4951";
    scale = "1 1 1";
    dataBlock = "SpawnSphereMarker";
    Radius = "1";
```

```
   sphereWeight = "100";
   indoorWeight = "100";
   outdoorWeight = "100";
      locked = "false";
      lockCount = "0";
      homingCount = "0";
};
new SpawnSphere() {
   canSaveDynamicFields = "1";
   position = "-154.957 897.763 194.53";
   rotation = "0 0 -1 2.47371";
   scale = "1 1 1";
   dataBlock = "SpawnSphereMarker";
   Radius = "1";
   sphereWeight = "100";
   indoorWeight = "100";
   outdoorWeight = "100";
      locked = "false";
      lockCount = "0";
      homingCount = "0";
};
new SpawnSphere() {
   canSaveDynamicFields = "1";
   position = "-167.979 897.871 194.986";
   rotation = "0 0 -1 2.47371";
   scale = "1 1 1";
   dataBlock = "SpawnSphereMarker";
   Radius = "1";
   sphereWeight = "100";
   indoorWeight = "100";
   outdoorWeight = "100";
      locked = "false";
      lockCount = "0";
      homingCount = "0";
};
new SpawnSphere() {
   canSaveDynamicFields = "1";
   position = "-143.463 929.037 192.885";
   rotation = "0 0 -1 10.4951";
   scale = "1 1 1";
   dataBlock = "SpawnSphereMarker";
   Radius = "1";
   sphereWeight = "100";
   indoorWeight = "100";
   outdoorWeight = "100";
      locked = "false";
```

```
      lockCount = "0";
      homingCount = "0";
};
new SpawnSphere() {
   canSaveDynamicFields = "1";
   position = "-156.097 928.417 194.47";
   rotation = "0 0 -1 2.47371";
   scale = "1 1 1";
   dataBlock = "SpawnSphereMarker";
   Radius = "1";
   sphereWeight = "100";
   indoorWeight = "100";
   outdoorWeight = "100";
      locked = "false";
      lockCount = "0";
      homingCount = "0";
};
new SpawnSphere() {
   canSaveDynamicFields = "1";
   position = "-169.119 928.525 194.926";
   rotation = "0 0 -1 2.47371";
   scale = "1 1 1";
   dataBlock = "SpawnSphereMarker";
   Radius = "1";
   sphereWeight = "100";
   indoorWeight = "100";
   outdoorWeight = "100";
      locked = "False";
      lockCount = "0";
      homingCount = "0";
};
new TSStatic() {
   canSaveDynamicFields = "1";
   position = "-214.079 904.392 219.664";
   rotation = "0.0698498 0.0515514 0.996225 107.35";
   scale = "2 2 2";
   shapeName = "~/data/shapes/trees/treea_1.dts";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0.8 0.7 0.6 1";
};
new TSStatic() {
   canSaveDynamicFields = "1";
```

```
   position = "-220.328 882.012 219.614";
   rotation = "1 0 0 0";
   scale = "2 2 2";
   shapeName = "~/data/shapes/trees/treea_1.dts";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0.8 0.7 0.6 1";
};
new TSStatic() {
   canSaveDynamicFields = "1";
   position = "-211.945 935.638 209.513";
   rotation = "1 0 0 15.4698";
   scale = "1 1 1";
   shapeName = "~/data/shapes/trees/treea_1.dts";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0.8 0.7 0.6 1";
};
new TSStatic() {
   canSaveDynamicFields = "1";
   position = "-208.929 921.685 215.407";
   rotation = "1 0 0 13.178";
   scale = "2 2 2";
   shapeName = "~/data/shapes/trees/treea_2.dts";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0.8 0.7 0.6 1";
};
new TSStatic() {
   canSaveDynamicFields = "1";
   position = "-211.017 888.841 218.414";
   rotation = "-0.0582764 -0.831159 0.552973 14.4525";
   scale = "2 2 2";
   shapeName = "~/data/shapes/trees/treea_2.dts";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
```

```
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0.8 0.7 0.6 1";
};
new TSStatic() {
   canSaveDynamicFields = "1";
   position = "-218.984 892.544 220.526";
   rotation = "0 0 -1 73.9115";
   scale = "2 2 2";
   shapeName = "~/data/shapes/trees/treea_2.dts";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new TSStatic() {
   canSaveDynamicFields = "1";
   position = "-222.468 875.204 220.681";
   rotation = "-0.0747834 0.0624526 -0.995242 100.537";
   scale = "1.5 1.5 1.5";
   shapeName = "~/data/shapes/trees/treea_2.dts";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0.8 0.7 0.6 1";
};
new SimGroup(mines) {
   canSaveDynamicFields = "1";

   new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-210.523 609.267 159.193";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
   };
```

```
   new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-227.826 490.199 150.383";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
   };
   new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-365.791 832.802 145.892";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
   };
   new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-252.712 1025.47 160.782";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
   };
   new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-203.378 1054.6 167.281";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
```

```
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
  };
  new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-183.978 1037.4 178.681";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
  };
  new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-230.72 1067.86 161.661";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
  };
  new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-180.709 1072.15 165.822";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
  };
  new StaticShape() {
```

```
      canSaveDynamicFields = "1";
      position = "-151.887 1004.37 188.262";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
    };
    new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-166.289 974.153 193.296";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
    };
    new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-181.587 997.92 192.654";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
    };
    new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-166.66 1026.25 183.593";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
```

```
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-183.534 1115.14 160.325";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-171.029 952.964 194.325";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-149.376 972.562 193.363";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
```

```
      position = "-149.266 1040.55 179.463";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
   };
   new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-168.487 1087.95 166.449";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
   };
   new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-157.281 957.227 193.974";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
   };
   new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-259.039 1042.72 161";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
```

```
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-291.263 958.775 150.347";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-339.55 876.021 159.141";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-337.029 769.409 152.024";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-367.829 802.809 152.624";
```

```
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
    };
    new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-359.961 760.824 153.171";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
    };
    new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-392.632 812.517 152.041";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
    };
    new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-347.054 692.584 153.191";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
```

```
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-395.117 704.37 163.412";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-393.18 669.009 173.301";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-362.321 631.187 158.689";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-357.346 657.456 157.771";
   rotation = "1 0 0 0";
```

```
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-328.43 590.406 161.764";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-349.621 562.706 161.86";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-322.049 540.061 157.873";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
```

```
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-356.445 501.875 161.819";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-314.522 479.776 152.594";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-282.465 504.292 152.198";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-207.972 532.243 149.775";
   rotation = "1 0 0 0";
   scale = "1 1 1";
```

```
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
    };
    new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-222.525 546.052 149.796";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
    };
    new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-188.954 823.207 194.361";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
    };
    new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-218.374 806.066 191.652";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
```

```
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-232.737 818.41 202.233";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-360.473 491.003 166.835";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-230.985 587.281 149.194";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-190.766 641.753 167.675";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
```

```
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-211.287 675.761 179.053";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-183.989 731.048 191.23";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-141.603 865.159 196.179";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "narki";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
```

```
   new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-116.973 883.108 192.929";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
   };
   new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-120.07 924.4 194.337";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "narki";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
   };
};
new SimGroup(health) {
   canSaveDynamicFields = "1";

   new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-200.215 477.872 150.312";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "healthkit";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
   };
   new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-126.769 901.041 193.714";
```

```
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "healthkit";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-227.325 833.554 203.27";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "healthkit";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-224.33 572.248 147.249";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "healthkit";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-362.462 504.334 164.647";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "healthkit";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
```

```
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-318.39 563.545 160.651";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "healthkit";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-407.167 664.018 182.86";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "healthkit";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-373.03 822.006 147.927";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "healthkit";
   receiveSunLight = "1";
   receiveLMLighting = "1";
   useAdaptiveSelfIllumination = "0";
   useCustomAmbientLighting = "0";
   customAmbientSelfIllumination = "0";
   customAmbientLighting = "0 0 0 1";
};
new StaticShape() {
   canSaveDynamicFields = "1";
   position = "-186.85 993.131 194.804";
   rotation = "1 0 0 0";
```

```
      scale = "1 1 1";
      dataBlock = "healthkit";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
    };
    new StaticShape() {
      canSaveDynamicFields = "1";
      position = "-188.024 1121.96 159.691";
      rotation = "1 0 0 0";
      scale = "1 1 1";
      dataBlock = "healthkit";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
    };
  };
};
//--- OBJECT WRITE END ---
```

Βιβλιογραφία – Ιστοσελίδες

- 3D Game Programming all in one - Kenneth Finney
- Advanced 3D Game Programming all in one - Kenneth Finney
- Frustrating the user on purpose: a step toward building an affective computer – Jocelyn Scheirer, Raul Fernandez, Jonathan Klein, Rosalind W. Picard
- Facial Expessions as Indicators of Appraisal Processes – Susanne Kaiser, Thomas Wehrle, University of Geneva
- Cognition and Emotion - Carien M. van Reekum, Tom Johnstone, Rainer Banse, Alexandre Etter, Thomas Wehrle, Klaus R. Scherer, University of Geneva
- Frustration Theory – Amsel, Cambridge University Press
- Inferring psychological significance from physiological signals – Cacioppo J, Tassinary L, American Psychologist
- Torque Game Engine Documentation
- GarageGames (http://www.garagegames.com)
- Hall of Worlds (http://www.hallofworlds.com)