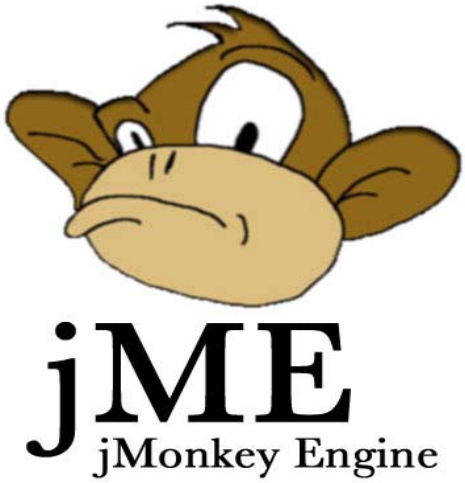
ΑΝΩΤΑΤΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ
ΙΔΡΥΜΑ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΠΟΛΥΜΕΣΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ



« ΑΝΑΠΤΥΞΗ ΤΡΙΣΔΙΑΣΤΑΤΟΥ ΔΙΑΔΡΑΣΤΙΚΟΥ ΠΑΙΧΝΙΔΙΟΥ ΠΕΡΙΗΓΗΣΗΣ»

ΕΙΡΗΝΗ ΚΥΡΙΑΚΙΔΟΥ Α.Μ. 427

Επιβλέπων Καθηγητής : κ. Μαμάκης Γιώργος

Ηράκλειο 2007

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1.

Πρόλογος

Η εργασία αυτή στοχεύει στην δημιουργία ένας τρισδιάστατου κόσμου αλληλεπίδρασης με ιδιότητες φυσικής (βαρύτητα, συγκρούσεις, τριβή κτλ) γραμμένο σε Java. Για να αναπτυχθεί μία τέτοιου είδους εφαρμογή εκτός από τον compiler της java ήταν απαραίτητη και η χρήση μιας 3D μηχανής, το jME.

Για την εφαρμογή μας χρησιμοποιήσαμε της εκδόσεις Java1.5 και NetBeans 5.0 Επίσης κάναμε εκτενή χρήση και άλλων βιβλιοθηκών όπως η LWGJL, η ODE και η odejava στις οποίες θα αναφερθούμε πιο αναλυτικά στην συνέχεια.

Εισαγωγή

Το jME (jMonkey Engine) είναι μια open source μηχανή υψηλής απόδοσης γραφικών. Ένα μεγάλο μέρος της έμπνευσης για το jME προέρχεται από το βιβλίο 3D Game Engine Design του David Eberly. Το jME φτιάχτηκε για να καλύψει την έλλειψη μιας πλήρης μηχανής γραφικής αναπαράστασης γραμμένη σε java.

Χρησιμοποιώντας ένα στρώμα αφαίρεσης, επιτρέπει σε οποιοδήποτε σύστημα απόδοσης (rendering system) να συνδεθεί. Αυτήν την περίοδο, υποστηρίζεται η LWJGL με προοπτική για την υποστήριξη της JOGL στο εγγύς μέλλον.

Το jME δημιουργήθηκε από τον Mark Powell το 2003 ενώ ερευνούσε το rendering του OpenGL. Όταν ανακάλυψε το LWJGL αποφάσισε ότι η java (που ήταν η γλώσσα της επιλογής του) θα ήταν τέλεια για τα δικά του εργαλεία γραφικής παράστασής. Αυτά τα εργαλεία σύντομα εξελίχθηκαν σε μια πρωτόγονη μηχανή. Μετά την ανάγνωση του βιβλίου του David Eberly, 3D Game Engine Design πραγματοποίησε μια αρχιτεκτονική γραφικών παραστάσεων σκηνής. Τότε ήταν, που και το jME έγινε μέρος του λογισμικού της Sun Java.net

Σύντομα προστέθηκαν πολλοί στην προσπάθεια για την ενίσχυση των δυνατοτήτων του jME. Από τότε τείνει να καλύψει πολλά προηγμένα και σύγχρονα χαρακτηριστικά γνωρίσματα γραφικής παράστασης και έχει εξελιχθεί πλέον σε μια σταθερή πλατφόρμα για την ανάπτυξη παιχνιδιών.

Ο Joshua Slack προστέθηκε στο δυναμικό του JME στο τέλος του 2003 και έγινε βασικό μέλος και ένα αναπόσπαστο κομμάτι της ομάδας του jME.

Game Engine (Μηχανή Παιχνιδιών)

Τι είναι όμως μια μηχανή παιχνιδιών; Μια μηχανή παιχνιδιών είναι ο πυρήνας ενός τμήματος λογισμικού από ένα video game ή μιας οποιασδήποτε άλλης εφαρμογής αλληλεπίδρασης πραγματικού χρόνου. Παρέχει τεχνολογίες οι οποίες απλοποιούν την δημιουργία και ανάπτυξη ενός

παιχνιδιού και συχνά επιτρέπει στις εφαρμογές μας, να έχουν την δυνατότητα να τρέχουν σε διαφορετικές πλατφόρμες, όπως οι κονσόλες παιχνιδιών ή τα λειτουργικά συστήματα των ηλεκτρονικών υπολογιστών Windows, Linux, Mac. Οι κύριες λειτουργίες που παρέχονται χαρακτηριστικά από μια μηχανή παιχνιδιών, είναι μια μηχανή απόδοσης (renderer) για δύο ή τριών διαστάσεων γραφική αναπαράσταση, μια μηχανή φυσικής ή ανίχνευσης σύγκρουσης, ήχος, τεχνητή νοημοσύνη, διαχείριση μνήμης, scripting, animation, networking, streaming, και μια σκηνή γραφικής αναπαράστασης.

Η βάση για κάθε μηχανή 3D βασίζεται στην μετατροπή των τριών διαστάσεων σε δυο. Η οθόνη του υπολογιστή μας είναι δύο διαστάσεων, πλάτους και ύψους. Με την βοήθεια πολλών μαθηματικών συναρτήσεων οι δημιουργοί των 3D μηχανών κατάφεραν να κάνουν αυτήν την μετατροπή εφικτή. Οι μηχανές αυτές εστιάζουν εκτός από το να μας δείχνουν πολύγωνα και στην ταχύτητα.

Για να αντισταθμιστούν οι χιλιάδες μαθηματικές πράξεις που πρέπει να γίνονται, αν υπολογιστεί ότι κάτι δεν θα είναι ορατό από κάποιο συγκεκριμένο σημείο, οι μηχανές παραβλέπουν αυτούς τους υπολογισμούς. Η διαδικασία αυτή έχει πάρει διάφορες ονομασίες, μια από αυτές είναι η "backface culling". Μια φόρμουλα η οποία είναι ευρέως διαδεδομένη στις περισσότερες μηχανές, είναι να χρησιμοποιείται από τους δημιουργούς των μηχανών ένα άπειρο επίπεδο για να ορίσει την ορατότητα αντικειμένων. Αν η φόρμουλα επιστρέψει την τιμή -1 τότε η μπροστινή πλευρά των αντικειμένων δεν φαίνεται. Βλέπουμε λοιπόν ότι ακόμα και όταν παραβλέπονται αντικείμενα πάλι οι μηχανές χρησιμοποιούν τα μαθηματικά.

Η φυσική είναι επίσης ζωτικής σημασίας στα παιχνίδια. Οι μηχανές παιχνιδιών μας επιτρέπουν να αλληλεπιδρούμε ζωντανά με τον κόσμο μας μέσω των μηχανών φυσικής τους. Έτσι δεν μπορούμε για παράδειγμα να περάσουμε μέσα από τους τοίχους ή αν πέσουμε από ένα ύψωμα θα συνεχίσουμε να πέφτουμε έως ότου φτάσουμε στο έδαφος. Η πτώση αυτή ανάλογα με το ύψος έχει και κάποιες συνέπειες. Όλοι αυτοί οι μαθηματικοί υπολογισμοί γίνονται σε πραγματικό χρόνο από το game engine.

Τα game engines παρέχουν εκτός από τα επαναχρησιμοποιήσιμα τμήματα λογισμικού και ένα σύνολο από οπτικά εργαλεία ανάπτυξης. Αυτά τα

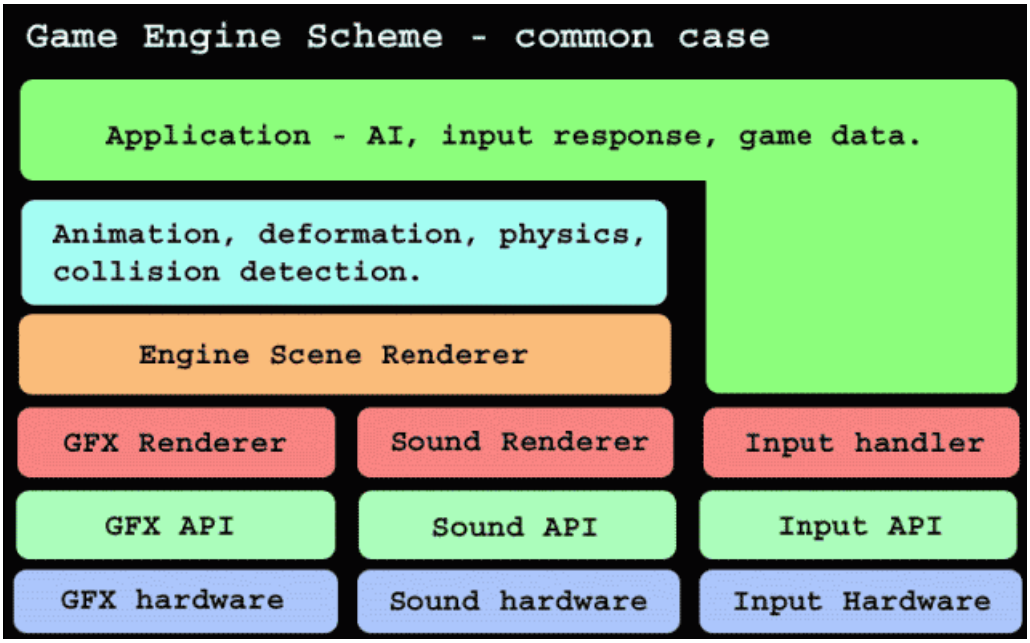
εργαλεία παρέχονται γενικά σε ένα ενσωματωμένο περιβάλλον ανάπτυξης, επιτρέποντας έτσι την απλουστευμένη και γρήγορη ανάπτυξη των παιχνιδιών.

Κάποιες μηχανές παιχνιδιών παρέχουν μόνο δυνατότητες 3D αναπαράστασης σε πραγματικό χρόνο αντί για ένα ευρύ φάσμα λειτουργιών που απαιτείται από τα παιχνίδια. Αυτές οι μηχανές βασίζονται στον δημιουργό του κάθε παιχνιδιού στο να εφαρμόσει τις υπόλοιπες λειτουργίες ή να τις συγκεντρώσει από άλλα τμήματα ήδη υπάρχον παιχνιδιών. Αυτοί οι τύποι μηχανών αναφέρονται ως μηχανές γραφικής παράστασης (graphics engines), μηχανές αναπαράστασης (rendering engine) ή μηχανές 3Δ (3D engine), αντί του μηχανές παιχνιδιών (game engine). Οι σύγχρονες μηχανές παιχνιδιών ή γραφικής παράστασης παρέχουν μια σκηνή αναπαράστασης (scene graph) η οποία είναι μια αντικειμενοστραφής αντιπροσώπευση του τρισδιάστατου κόσμου παιχνιδιών που απλοποιεί συχνά το σχέδιο παιχνιδιών και μπορεί να χρησιμοποιηθεί για την καλύτερη απόδοση των απέραντων εικονικών κόσμων.

Ο όρος “ μηχανή παιχνιδιών” προέκυψε στα μέσα της δεκαετίας του 90 ειδικά για τα 3D παιχνίδια όπως τα first person shooters (FPS).Παιχνίδια όπως το Doom και το Quake δεν δημιουργήθηκαν από το μηδέν. Κάποιοι προγραμματιστές είχαν χορηγήσει άδειες για σημαντικά τμήματα του λογισμικού και έτσι έπρεπε να σχεδιαστούν μόνο οι γραφικές σκηνές, οι χαρακτήρες , τα όπλα , τα επίπεδα, καθώς και τα περιεχόμενα και προτερήματα των παιχνιδιών.

Απεικόνιση 3D μηχανών

Όπως έχουμε ήδη αναφέρει η μηχανή παιχνιδιών (ή 3D μηχανή) είναι ένα πολύπλοκο σύστημα, υπεύθυνο για τον σχηματισμό της εικόνας και του ήχου ενός παιχνιδιού που χειρίζεται τα δεδομένα που εισάγει ο χρήστης και παρέχει πηγές διαχείρισης, animation (απεικόνιση με κινούμενα σχέδια), φυσική και πολλά άλλα.



Στο χαμηλότερο επίπεδο βρίσκεται το hardware. Το API το οποίο έχει πρόσβαση στο hardware βρίσκεται ένα επίπεδο πιο πάνω. Συχνά οι τρισδιάστατες μηχανές ή τα συστήματα αναπαράστασης σε μια μηχανή παιχνιδιών είναι χτισμένες πάνω σε ένα API γραφικών όπως το Direct3D ή το OpenGL που ελαφρύνουν κατά κάποιο τρόπο το λειτουργικό του GPU (Graphics Processing Unit) ή της τηλεοπτικής κάρτας. Οι χαμηλού επιπέδου βιβλιοθήκες όπως DirectX, OpenAL επίσης χρησιμοποιούνται συνήθως στα παιχνίδια καθώς παρέχουν ανεξαρτησία hardware παρέχοντας πρόσβαση και σε άλλους υπολογιστές με διαφορετικά χαρακτηριστικά μονάδων εισόδου (ποντίκι, πληκτρολόγιο, joystick), κάρτες δικτύου και ήχου. Η εισαγωγή δεδομένων γίνεται μέσω του DirectInput. Το GFX Renderer είναι υπεύθυνο για την απεικόνιση της τελικής σκηνής με όλα τα φαντασμαγορικά εφφέ. Το Sound Renderer έχει την ίδια θέση με τον ήχο και παίζει τους ήχους την σωστή στιγμή χρησιμοποιώντας τα σωστά εφφέ. Ο Input Handler συγκεντρώνει πληροφορίες και αλλαγές από το πληκτρολόγιο, το ποντίκι ή οποιεσδήποτε άλλες μονάδες εισόδου έχουμε και τις μετατρέπει σε τέτοια μορφή που να γίνονται αποδεκτές από την μηχανή. Είναι επίσης υπεύθυνο για να δίνει σωστές εντολές πίσω στις μονάδες (devices) ανάλογα με τις συνέπειες που ίσως υπάρχουν. Το engine scene renderer χρησιμοποιεί μεθόδους χαμηλότερου επιπέδου για να αποδώσει την σκηνή πάνω στην οθόνη και για να ακουστούν οι σωστοί ήχοι. Το επίπεδο πάνω από τον

Renderer έχει πολλές λειτουργίες που συνεργάζονται μεταξύ τους. Συμπεριλαμβάνει animation και collision detection. Το Deformation χρησιμοποιεί τη φυσική για να προσδιορίσει τα σχήματα σε σχέση με τις δυνάμεις που ασκούνται. Μέσω της φυσικής μετριέται και υπολογίζεται η βαρύτητα, ο άνεμος κ.α. Σε αυτό το επίπεδο μπορεί να υπάρχουν και πολλές άλλες λειτουργίες οι οποίες εξαρτώνται από την εκάστοτε εφαρμογή. Πάνω από όλα αυτά βρίσκεται η εφαρμογή μας η οποία είναι υπεύθυνη για τα δεδομένα του παιχνιδιού, το AI (Artificial Intelligence), το GUI (Graphical User Interface) και επίσης τα δεδομένα που εισάγει ο χρήστης.

Οι σύγχρονες μηχανές παιχνιδιών είναι μερικές από τις πιο σύνθετες εφαρμογές που έχουν γραφτεί χαρακτηρίζοντας συχνά πληθώρα από άριστα συντονισμένα συστήματα αλληλεπίδρασης για να εξασφαλίσουν μια πλήρης ελεγχόμενη εμπειρία για τους χρήστες. Η συνεχής βελτίωση των μηχανών παιχνιδιών έχει δημιουργήσει έναν ισχυρό διαχωρισμό μεταξύ του rendering, scripting,artwork και το level design.

Τα παιχνίδια First Person Shooter παραμένουν οι κυρίαρχοι των μηχανών παιχνιδιών 3D. Το threading όσο περνάει ο καιρός παίρνει περισσότερη σημασία λόγο των σύγχρονων multi –core συστημάτων και τις αυξημένες απαιτήσεις στην ρεαλιστικότητα. Τα threads χαρακτηριστικά περιλαμβάνουν το rendering, το streaming, τον ήχο και την φυσική.

Σε κάθε γενιά των 3D accelerators βλέπουμε ομορφότερα και πιο ρεαλιστικά παιχνίδια με περισσότερες αλληλεπιδράσεις και πιο περίπλοκα μοντέλα.

LWGJL (Lightweight Java Gaming Library)

Η ελαφριά βιβλιοθήκη παιχνιδιών της Java (LWJGL) είναι μια λύση που στοχεύει άμεσα στους επαγγελματίες και ερασιτέχνες προγραμματιστές και τους επιτρέπει να δημιουργούν παιχνίδια εμπορικής ποιότητας γραμμένα σε java. Η LWJGL παρέχει πρόσβαση υψηλής απόδοσης σε crossplatform libraries (παίζουν όπου μπορεί να παίξει java) όπως η OpenGL (Open

Graphics Library) και η OpenAL (Open Audio Library) . Επιπλέον, η LWJGL παρέχει πρόσβαση σε controllers όπως Gamepads, Steering wheel και Joysticks όλα αυτά μέσα από ένα εύκολο και απλό API.

Η LWJGL δεν καθιστά ιδιαίτερα εύκολο το γράψιμο των παιχνιδιών. Είναι πρωτίστως μια τεχνολογία η οποία επιτρέπει στους προγραμματιστές να αναπτύξουν πόρους οι οποίοι διαφορετικά είναι ή μη διαθέσιμοι ή κακώς σχεδιασμένοι στην υπάρχουσα java. Προσδοκάται ότι η LWJGL, μέσω της εξέλιξης και της επέκτασης της, θα αποτελέσει θεμέλιο για τις πληρέστερες βιβλιοθήκες παιχνιδιών ή “τις μηχανές παιχνιδιών” όπως είναι ευρύτερα γνωστό..

OpenGL (Open Graphics Library)

Η OpenGL είναι το αρχαιότερο περιβάλλον που καθορίζει ένα cross-language API για την ανάπτυξη φορητών, διαλογικών 2D και 3D εφαρμογών γραφικής παράστασης. Η OpenGL αναπτύχθηκε από την Silicon Graphics Inc. (SGI) το 1992 και είναι δημοφιλής στην βιομηχανία των video παιχνιδιών όπου ανταγωνίζεται το Direct3D της Microsoft.

Το interface της αποτελείται πάνω από 250 διαφορετικές λειτουργίες που μπορούν να χρησιμοποιηθούν για να δημιουργήσουν σύνθετες τρισδιάστατες σκηνές από απλά σχήματα. Η OpenGL χρησιμοποιείται ευρέως στο CAD, την εικονική πραγματικότητα, την επιστημονική απεικόνιση, την απεικόνιση πληροφοριών, την προσομοίωση πτήσης και την τηλεοπτική ανάπτυξη παιχνιδιών.

Στο πιο βασικό της επίπεδο η OpenGL είναι ένα έγγραφο το οποίο περιγράφει ένα σύνολο λειτουργιών και τις ακριβής συμπεριφορές που πρέπει να εκτελούνται.

Η OpenGL εξυπηρετεί δύο κύριους σκοπούς:

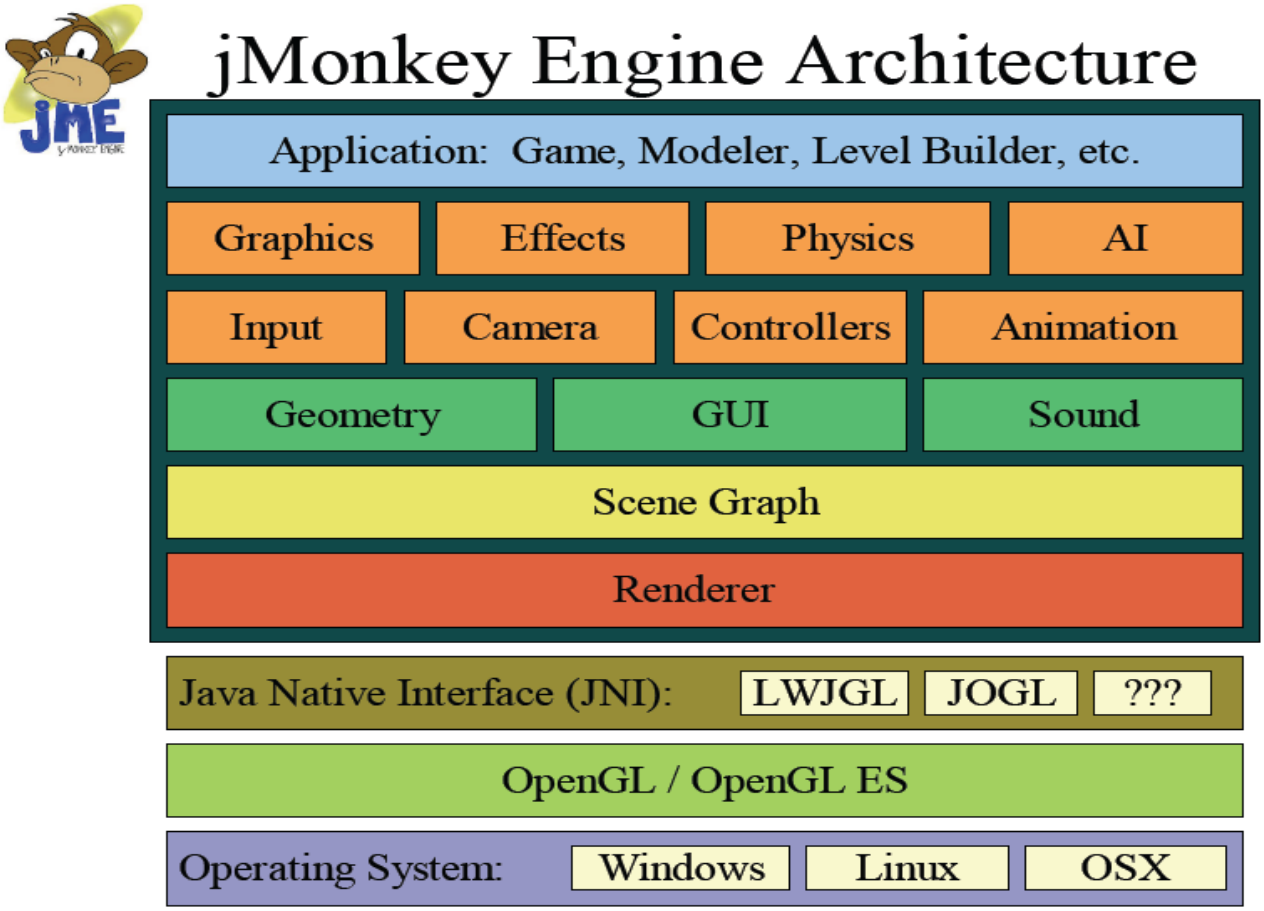
1. Κρύβει την πολυπλοκότητα διασύνδεσης με τους διαφορετικούς 3D acceleratos παρέχοντας ένα ενιαίο και ομοιόμορφο API στον προγραμματιστή.

2. Κρύβει τις διαφορετικές ικανότητες των πλατφόρμων υλικού, με την απαίτηση ότι όλες οι εφαρμογές πρέπει υποστηρίζουν το πλήρες σύνολο χαρακτηριστικών γνωρισμάτων της OpenGL

Βασική λειτουργία της openGL είναι να γίνουν αποδεκτά απλά στοιχεία όπως σημεία, γραμμές και πολύγωνα, και να το μετατραπούν σε Pixels. Αυτό γίνεται με το graphics pipeline (τρόπος με τον οποίο γίνονται αποδεκτές εισαγωγές 3D δεδομένων) που είναι γνωστό ως OpenGL state machine. Οι περισσότερες εντολές στην OpenGL δίνουν στο graphics pipeline απλά σχήματα η συντονίζουν πώς το pipeline θα επεξεργαστεί τα σχήματα αυτά.

Αρχιτεκτονική του jMonkey Engine.

Τα επίπεδα αρχιτεκτονικής του jME απεικονίζονται στο παρακάτω σχήμα.



Αν και όλα τα επίπεδα θα τα δούμε στην συνέχεια αναλυτικά, συνοπτικά μπορούμε να πούμε ότι στο πρώτο επίπεδο ορίζονται τα λειτουργικά συστήματα στα οποία μπορούν να τρέχουν οι εφαρμογές του jME. Ας μην ξεχνάμε ότι το jME είναι 100% Java και εξαρτάται από πλατφόρμα JNI (Java Native Interface). Έτσι τα λειτουργικά συστήματα τα οποία μπορούμε να χρησιμοποιήσουμε είναι τα Windows, Linux και OSX όπου μπορεί να τρέχει η LWJGL.

Στο επόμενο στάδιο βρίσκονται τα OpenGL/ OpenGL ES. Όπως ήδη αναφέραμε η OpenGL είναι ένα cross –platform API για υψηλή απόδοση 2D και 3D γραφικών. Η OpenGL ES παρέχει προχωρημένες δυνατότητες για 2D/3D γραφικά, μεγάλη ποικιλία από κινητές συσκευές και συμπτυγμένες απεικονίσεις.

Στο επόμενο επίπεδο βρίσκεται το JNI. Η τεχνική JNI (Java Native Interface) είναι η τοπική διασύνδεση προγραμμάτων της Java το οποίο είναι μέρος του JDK. Η τεχνική JNI επιτρέπει τον κώδικα της Java, ο οποίος τρέχει με ένα φανταστικό μηχάνημα της Java VM (Virtual Machine), να χειρίζεται εφαρμογές και βιβλιοθήκες οι οποίες είναι γραμμένες σε άλλες γλώσσες, όπως είναι η C, C++ και assembly.

Αμέσως πιο πάνω βρίσκεται ο Renderer. Οι ιδιότητες του Renderer είναι να μετατρέπει τα δεδομένα της σκηνής από τον πραγματικό κόσμο στο χώρο της οθόνης. Επίσης για να έχουμε καλύτερη απόδοση σε ταχύτητα, αφαιρεί από την διαδικασία του Rendering κομμάτια από την σκηνή παράστασης με τους ακόλουθους τρόπους :

- Culling : είναι η διαδικασία κατά την οποία διευκρινίζεται αν κάποιο από τα μέρη ενός αντικειμένου είναι ορατό από κάποιο συγκεκριμένο σημείο αναφοράς. Αν όχι τότε δεν θα περάσει από την διαδικασία του Render.
- Clipping : είναι η διαδικασία κατά την οποία χωρίζεται ένα αντικείμενο σε μικρότερα κομμάτια και τα κομματάκια τα οποία δεν είναι ορατά δεν περνάν από rendering.

Επίσης ο Renderer είναι υπεύθυνος για την αναπαράσταση της μορφοποιημένης- τροποποιημένης σκηνής μας στην οθόνη.

Ακολουθεί το scene graph. Το scene graph μπορούμε να το παρομοιάσουμε με ένα δέντρο όπου τα δεδομένα είναι κατηγοριοποιημένα σύμφωνα με μια ιεραρχική δομή. Η δομή αυτή περιέχει:

- Τους κόμβους γονέα (parent Nodes) που περιέχουν όσους κόμβους παιδιών θέλουμε.
- Τους κόμβους παιδιών (child Nodes) που έχουν έναν μόνο parent Node.
- Το rootNode που δεν έχει κανένα parent Node.
- Τα Leaf Nodes που περιέχουν γεωμετρικά δεδομένα
- Εσωτερικούς κόμβους για την διαχείριση ομάδων.

Πλεονεκτήματα scene graph:

1. απλοποιεί την διαχείριση γενικών χαρακτηριστικών π.χ. ορίζει φως για να φωτίσει ένα μόνο υποδέντρο της σκηνής
2. επιτρέπει την ομαδοποίηση αντικειμένων που βρίσκονται στην ίδια χωροταξική περιοχή π.χ. βοηθάει στην γρήγορη απαλοιφή περιοχών της σκηνής οι οποίες δεν θα είναι ορατές και έτσι δεν χρειάζονται να περάσουν από την διαδικασία του rendering.
3. διευκολύνει τον προσανατολισμό των ιεραρχικών μοντέλων όπως οι ανθρωποειδής χαρακτήρες π.χ. το κορμί καθορίζει την θέση του κεφαλιού, των χεριών και των ποδιών. Τα χέρια προσδιορίζουν την θέση της παλάμης κτλ.

Η απεικόνιση του scene graph σαν δέντρο προσφέρει:

1. οι κόμβοι να διατηρούν σημαντικές πληροφορίες και πληροφορίες χώρου.
2. Μετασχηματισμούς, για την αλλαγή θέσης, προσανατολισμού και μεγέθους των αντικειμένων.

3. Bounding Volumes, χρησιμοποιούνται για την αφαίρεση κομματιών ιεραρχικά (hierarchal culling) και για τεστ συγκρούσεων.
4. Render State, χρησιμοποιείται για να στηθεί ο renderer ο οποίος θα απεικονίσει τα αντικείμενα.
5. Animation State, χρησιμοποιείται για να δείξει πως ο χρόνος επηρεάζει τα δεδομένα των κόμβων.

Στο αμέσως επόμενο στάδιο βρίσκεται η γεωμετρία, ο ήχος και το GUI. Η γεωμετρία προσδιορίζει τους Leaf κόμβους της σκηνής και περιέχουν γεωμετρικά δεδομένα που χρειάζονται για το Rendering των αντικειμένων. Επίσης διαχειρίζεται όλες τις πληροφορίες rendering συμπεριλαμβανομένου διαφορετικών καταστάσεων rendering, και δεδομένα των μοντέλων. Το GUI περιλαμβάνει τους Layout Managers. Οι ήχοι γίνονται render σε 3D χώρο και καθορίζονται από 2 interfaces. Το ISoundSystem που είναι υπεύθυνο για να παίζει την μουσική και το ISoundRenderer που διαχειρίζεται την σύνδεση του ήχου με την σκηνή αναπαράστασης καθώς και την διαχείριση των 3D ήχων.

Στο επόμενο επίπεδο βρίσκονται αρκετές λειτουργίες οι οποίες αλληλεπιδρούν μεταξύ τους. Οι δυνατότητες των γραφικών είναι πάρα πολλές όπως να κάνουμε render ένα texture, να έχουμε multi-texturing υποστήριξη, να έχουμε την δυνατότητα φόρτωσης διάφορων τύπων μοντέλου, δημιουργία απλών σχημάτων (σημείου, γραμμής, κουτιού, σφαίρας κτλ), διαχείρισης καταστάσεων όπως (lights, culling, texturing, Zbuffer ,Materials), δυνατότητα για εφφέ (μέχρι στιγμής διαθέσιμα είναι το tinting και το particle system (ίχνη). Από μονάδες εισόδου μέχρι στιγμής υποστηρίζονται το mouse και το πληκτρολόγιο. Η camera χρησιμοποιείται για να προβάλει τα αντικείμενα στην οθόνη του υπολογιστή. Είναι ένας συνδυασμός από την θέση του ματιού- τη θέση της cameras, το οπτικό πεδίο (view plane), το μέρος του οπτικού πεδίου που θα φαίνεται (view port) και το view frustrum. Οι controllers μεταλλάσσουν τους κόμβους και τα render states σε σχέση με το χρόνο.

Στο τελευταίο στάδιο βρίσκονται τα πρότυπα, τα παιχνίδια και γενικά όλες οι εφαρμογές μας που δημιουργούμε…

Χαρακτηριστικά γνωρίσματα του jME

Το jMonkey Engine σχεδιάστηκε να είναι μια μηχανή γραφικών μεγάλης ταχύτητας σε πραγματικό χρόνο. Με τις προόδους που έγιναν τόσο στα graphics hardware όσο και στην γλώσσα προγραμματισμού της java , η ανάγκη για μια βιβλιοθήκη παιχνιδιών βασισμένη στην java ήταν πλέον προφανής. Το jME καλύπτει πολλές από αυτές τις ανάγκες παρέχοντας ένα σύστημα υψηλής απόδοσης (rendering system) γραφικών παραστάσεων σκηνής.

Ο πυρήνας του JME είναι το scene graph. Το scene graph είναι η δομή δεδομένων που διατηρεί τα στοιχεία του κόσμου Οι σχέσεις μεταξύ των δεδομένων του παιχνιδιού (γεωμετρίας , ήχου, φυσικής) διατηρούνται σε μια δομή δέντρου, όπου οι κόμβοι φύλλων αντιπροσωπεύουν τα στοιχεία του πυρήνα του παιχνιδιού. Αυτά τα βασικά στοιχεία επί το πλείστον είναι αυτά που φαίνονται στη σκηνή, ή που ακούγονται μέσω της κάρτας ήχου. Η οργάνωση του scene graph είναι πολύ σημαντική και εξαρτάται από την εκάστοτε εφαρμογή. Γενικά το scene graph αντιπροσωπεύει ένα μεγάλο αριθμό δεδομένων που έχει χωριστεί σε μικρότερα εύκολοσυντηρήσιμα κομμάτια.

Αυτά τα κομμάτια ομαδοποιούνται με βάση κάποιο είδος σχέσης, συνηθέστερα από τη χωροταξική τους θέση. Αυτή η ομαδοποίηση επιτρέπει σε μεγάλα τμήματα δεδομένων του παιχνιδιών να αφαιρεθούν από την επεξεργασία εάν δεν απαιτούνται για να επιδείξουν την τρέχουσα σκηνή. Καθορίζοντας ότι ένα τμήμα του κόσμου δεν απαιτείται να υποβληθεί σε επεξεργασία, χρησιμοποιείται λιγότερη CPU και ξοδεύεται λιγότερος χρόνος GPU για την επεξεργασία των δεδομένων με αποτέλεσμα να βελτιώνεται η ταχύτητα του παιχνιδιού.

Το scenegraph επιτρέπει την οργάνωση των στοιχείων του παιχνιδιών σε μια δομή δέντρων, όπου ένας κόμβος γονέων (parent node) μπορεί να περιέχει όσους κόμβων παιδιών (children nodes) θέλει. Αλλά, ένας κόμβος παιδιών περιέχει μόνο έναν κόμβο γονέα. Συνήθως, αυτοί οι κόμβοι οργανώνονται στο χώρο για να επιτρέπουν τη γρήγορη απόρριψη ολόκληρων υποδιαιρέσεων για επεξεργασία.

Για παράδειγμα, έστω ότι έχουμε δημιουργήσει κάποια αντικείμενα τα οποία βρίσκονται μέσα σε ένα δωμάτιο. Όλα τα αντικείμενα μοιράζονται τον γονέα δωμάτιο , όλα τα δωμάτια μοιράζονται τον γονέα πάτωμα και όλα τα πατώματα έχουν τον γονέα κτίριο. Αν ο παίκτης είναι στο πρώτο δωμάτιο του πρώτου ορόφου τότε μπορούμε εύκολα και γρήγορα να απορρίψουμε τον κόμβο του δεύτερου ορόφου και αυτόματα να απορριφθούν και όλα τα δωμάτια που βρίσκονται στον όροφο αυτό, καθώς και όλα τα αντικείμενα που βρίσκονται μέσα σε αυτά τα δωμάτια. Μπορούμε έπειτα να επεξεργαστούμε το πάτωμα 1. Όλα τα δωμάτια που δεν είναι δωμάτιο 1 απορρίπτονται, καθώς και όλα τα αντικείμενα των δωματίων αυτών. Επεξεργαζόμαστε έπειτα το δωμάτιο 1 συμπεριλαμβανομένου όλων των αντικείμενων του.

Η απόρριψη – αφαίρεση, μπορεί να σημαίνει ποικίλα πράγματα, αλλά ο σημαντικότερος στον προγραμματισμό γραφικής αναπαράστασης είναι το ξεδιάλεγμα των δεδομένων(culling of data). Το σύστημα κάμερας του jME χρησιμοποιεί το frustum culling για να αφαιρεί κομμάτια της σκηνής που δεν είναι ορατά. Αυτό επιτρέπει στις σύνθετες σκηνές να αποδίδονται γρήγορα, αφού στις περισσότερες περιπτώσεις το μεγαλύτερο μέρος της σκηνής δεν είναι ορατό τον περισσότερο χρόνο

Ενώ η γραφική παράσταση σκηνής (scene graph) είναι ο πυρήνας των στοιχείων της γραφικής παράστασης του jME, τα εργαλεία εφαρμογής παρέχουν τα μέσα να δημιουργηθεί γρήγορα το πλαίσιο γραφικής παράστασης και να αρχίσει ένας κύριος βρόχος παιχνιδιών. Η δημιουργία του παραθύρου, το σύστημα εισαγωγής (input system), κάμερας και το σύστημα παιχνιδιών δεν είναι τίποτα περισσότερο από μια ή δύο κλήσεις της κάθε μεθόδου. Αυτό επιτρέπει στον προγραμματιστή να σταματήσει να χάνει το χρόνο του εξετάζοντας το σύστημα και να έχει περισσότερο χρόνο να ασχοληθεί με την εφαρμογή του.

Ο χρήστης απαλλάσσεται ουσιαστικά από το rendering του scene graph. Το όφελος αυτό μας επιτρέπει την ανταλλαγή σε διαφορετικά συστήματα απόδοσης χωρίς να χρειάζεται να αλλαχτεί ούτε μια γραμμής κώδικα. Παραδείγματος χάριν, εάν τρέχουμε την εφαρμογή μας χρησιμοποιώντας την LWJGL μπορούμε να μεταπηδήσουμε εύκολα σε

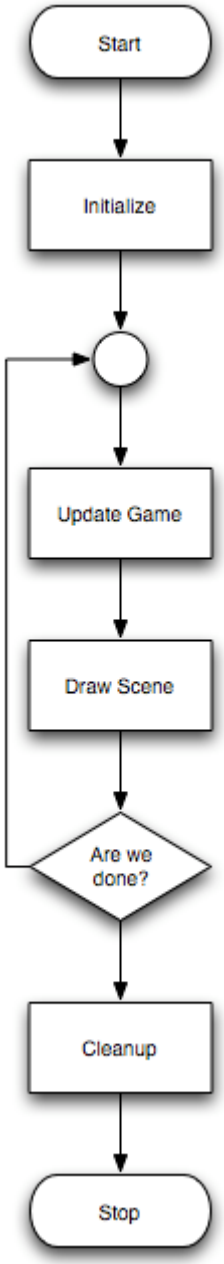
JOGL της sun. Το τελικό αποτέλεσμα θα διαφέρει ελάχιστα και δεν θα χρειάζεται να ξαναγραφτεί το πρόγραμμα μας. Εντούτοις, για να δημιουργηθεί ένα καλύτερο API, η εστίαση μέχρι τώρα είναι στην προσθήκη νέων χαρακτηριστικών γνωρισμάτων παρά τη δημιουργία ενός νέου συστήματος απόδοσης . Επομένως, η LWJGL είναι αυτήν την περίοδο το μόνο σύστημα renderer που υποστηρίζεται. (το JOGL έχει δημιουργηθεί αλλά δεν υποστηρίζεται μέχρι στιγμής)

ΚΕΦΑΛΑΙΟ 2.

Main Game Loop

Στον πυρήνα κάθε παιχνιδιού υπάρχει ένας βρόχος. Ο βρόχος αυτός είναι ένα τμήμα κώδικα που εκτελείται συνέχεια καθ' όλη την διάρκεια του παιχνιδιού και είναι υπεύθυνος για να συντονίζει έναν κύκλο από ενημερώσεις και οι απεικονίσεις (update/draw). Όλες οι AbstactGame υποκλάσεις παίζουν ρόλο στην δημιουργία αυτού του βρόχου.

Παρ΄ όλο που οι υποκλάσεις ποικίλουν όλες ακολουθούν την λογική που φαίνεται στο παρακάτω σχήμα. Για την δημιουργία των σχημάτων στης εφαρμογής μας χρειαζόμαστε μια main class που θα περιέχει την μέθοδο main. Το jme παρέχει την κλάση AbstractGame και τις υποκλάσεις της. Η AbstractGame παρέχει την main game loop. Αυτός ο βρόχος αποτελεί το στήριγμα της εφαρμογής μας γιατί αυτός είναι υπεύθυνος για την καθοδήγηση και ολοκλήρωση της εφαρμογής μας. Καλείται επαναλαμβανόμενα έως ότου η εφαρμογή μας τελειώσει (επανάληψη ανά frame) Το πώς θα διεξαχθεί το game loop εξαρτάται από ποιες υποκλάσεις του AbstractGame θα επιλέξουμε. Το game loop ξεκινάει όταν καλέσουμε την μέθοδο start



PropertiesDialog

Το PropertiesDialog είναι ένα dialog το οποίο εμφανίζεται -αν το επιλέξουμε εμείς - όταν ξεκινάει η εφαρμογή μας και ζητά πληροφορίες για το πως θα τρέξει η εφαρμογή μας (display settings). Καλώντας την μέθοδο setDialogBehaviour Το dialog μας δίνει την δυνατότητα να επιλέξουμε την ανάλυση(resolution), το βάθος χρώματος(color depth), αν θα είναι full screen

και τον ρυθμό ανανέωσης(refresh rate) και την δυνατότητα επιλογής API. Έπειτα αυτές οι ιδιότητες αποθηκεύονται σε ένα configuration file με την βοήθεια του PropertiesIO κάθε φορά που πάμε να εκτελέσουμε την εφαρμογή μας . Ανάλογα με το τι παραμέτρους θα δώσουμε στην setDialogBehaviour, το dialog θα εμφανίζεται πάντα, δεν θα εμφανίζεται ποτέ, ή θα εμφανίζεται όταν δεν υπάρχει το properties configuration file. Η AbstractGame δημιουργεί από μόνη της το dialog αυτό και εμείς μπορούμε να έχουμε πρόσβαση σε αυτήν απλά καλώντας την κλάση.

Ανάλυση παραμέτρων του Properties dialog:

NEVER_SHOW_PROPS_DIALOG. Δεν θα εμφανιστεί ποτέ το παράθυρο επιλογών και έτσι δεν θα υπάρχει το properties file. Με αυτήν την επιλογή πρέπει να είμαστε σίγουροι το σύστημα μας έχει την δυνατότητα να τρέξει από default εφαρμογές με χαρακτηριστικά 640x480x60Hzx16bits fullscreen.

FIRSTRUN_OR_NOCONFIGFILE_SHOW_PROPS_DIALOG . θα εμφανιστεί το παράθυρο επιλογών αν δεν υπάρχει το configuration file. Έτσι επιτρέπετε στον χρηστή να αλλάξει της παραμέτρους που θέλει μια φορά στην αρχή και να μην χρειάζεται να ξανα ασχοληθει με αυτό το θέμα. Ωστόσο αν ο χρηστής θελήσει για κάποιο λόγο αργότερα να αλλάξει τις παραμέτρους αυτές θα πρέπει να σβήσει το properties file έτσι ώστε να μην υπάρχει για να του ξαναζητηθεί από το σύστημα να τα ορίσει από την αρχή.

ALWAYS_SHOW_PROPS_DIALOG . πάντα θα εμφανίζεται το παράθυρο επιλογών δίνοντας την δυνατότητα για εύκολη και γρήγορη αλλαγή των παραμέτρων.

Το PropertiesIO όπως είπαμε και προηγουμένως δημιουργεί ένα properties file και κρατάει αποθηκευμένα τις ρυθμίσεις της εφαρμογής μας. Εμείς δεν χρειάζεται να κάνουμε κάτι ιδιαίτερο αφού η AbstractGame ορίζει από μόνη της την κλάση PropertiesIO και αποθηκεύει το αρχείο αυτό ως properties.cfg στο τρέχον directory.

Υπάρχουν αρκετοί τύποι παιχνιδιών που παρέχει η βιβλιοθήκη του JME. Εμείς θα αναφερθούμε στην BaseGame, SimpleGame που είναι και οι

πιο βασικές. Η SimplePhysicsGame που είναι γραμμένη η εφαρμογή μας είναι υποκλάση της BaseGame.

BaseGame.

Η BaseGame παρέχει την πιο βασική εφαρμογή της κλάσης AbstractGame. Αυτό που κάνει είναι και καθορίζει το main game loop και μόνο. Το loop είναι ταχύτατο και κάθε του επανάληψη εξαρτάται από το πόσο γρήγορα δουλεύουν οι CPU/GPU. Σε αυτήν την περίπτωση είμαστε υποχρεωμένοι να δημιουργήσουμε τις παρακάτω μεθόδους.

InitSystem. Με την μεθοδο αυτή στήνουμε το display system.Τα στοιχεία που περιέχει(κάμερα, input system κτλ) δεν είναι στοιχεία του παιχνιδιού και εδώ γίνεται η αρχικοποίηση τους.

InitGame. Είναι η μέθοδος όπου δημιουργούνται όλα τα δεδομένα του παιχνιδιού και των γραφικών.

Update. Με την μέθοδο αυτή ανανεώνουμε τα δεδομένα σε κάθε παιχνίδι. Αυτό μπορεί να είναι ο timer του παιχνιδιού, μέθοδοι εισαγωγής δεδομένων, έλεγχος συγκρούσεων (collision checks). Οτιδήποτε μεταβάλλεται στην διάρκεια του χρόνου θα πρέπει να περνάει μέσα από την μέθοδο αυτή.

Render. Η μέθοδος αυτή διαχειρίζεται το πώς θα εμφανιστεί γραφικά η σκηνή(scene).

Reinit. Είναι η μέθοδος που χρησιμοποιούμε όταν το σύστημα μας χρειάζεται να ξαναδημιουργήσει κάτι (rebuild) π.χ. όταν αλλάζουμε τις ρυθμίσεις στο display.

Cleanup. Η μέθοδος αυτή κλείνει τυχών ανοιχτές πήγες που έχουμε αφήσει έτσι ώστε να επιτρέψει στο σύστημα να επιλύσει το πρόβλημα ομαλά.

SimpleGame.

Ένας άλλος τύπος για γρήγορο και εύκολο σχεδιασμό παιχνιδιών είναι η SimpleGame. Η SimpleGame είναι υποκλάση της BaseGame και “κληρονομεί” όλες τις Abstract μεθόδους της BaseGame. Στην SimpleGame υπάρχει εξ’ ορισμού ένα root scene node (rootNode) όπου μπορούν να προσκολληθούν όλα τα δεδομένα του παιχνιδιού. Σε αυτόν τον γονικό κόμβο γίνετε αυτόματα render. Όλα τα υπόλοιπα αρχικοποιούνται από μόνα τους συμπεριλαμβανόμενου του συστήματος εισαγωγής δεδομένων (FirstPersonHandler), του timer, η απόδοση του : lighting, zbuffer, και WireFrameState (εξ ορισμού είναι ανενεργό).

Επίσης περιέχει την μέθοδο simpleInitGame στην οποία ο χρηστής δημιουργεί όλα τα στοιχεία και τα δεδομένα που θέλει να έχει η εφαρμογή του. Όλα αυτά παρέχουν μια βασική πλατφόρμα έτσι ο χρηστής να μπορεί να πειραματιστεί με τα χαρακτηρίστηκα του JME χωρίς να χρειάζεται να ανησυχεί για το πώς θα στήσει μια πλήρης σε χαρακτηρίστηκα εφαρμογή.

Αν θέλουμε μπορούμε να κάνουμε override την simpleUpdate για να έχουμε επιπρόσθετες απαιτήσεις ανανέωσης. Αυτό θα προσθέσει μόνο επιπλέον ιδιότητες στην update της SimpleGame χωρίς να καταργεί κάποια ήδη υπάρχουσα. Επίσης μπορούμε να κάνουμε override στην simpleRender ούτε και σε αυτήν την περίπτωση καταργείται κάτι που ήδη υπάρχει.

Η SimpleGame δημιουργεί ένα σύστημα εισαγωγής δεδομένων ΄με βάση τα παρακάτω πλήκτρα

Key	Action
W	Move Forward
A	Strafe Left
S	Move Backward
D	Strafe Right
Up	Look Up
Down	Look Down

Left Look Left

Right Look Right

T Wireframe Mode on/off

P Pause On/Off

L Lights On/Off

C Print Camera Position

B Bounding Volumes On/Off

N Normals On/Off

F1 Take Screenshot

Πρόσβαση στον InputHandler μπορούμε να έχουμε χρησιμοποιώντας την μεταβλητή input που παρέχει η SimpleGame

Η SimpleGame δημιουργεί μια στάνταρ camera στην θέση (0,0,25) , αριστερό διάνυσμα (-1,0,0), πάνω (0,1,0) και διεύθυνση (0,0,-1). Στην simpleGame αυτή η camera αναφέρεται ως cam και ο χειρισμός της γίνεται από την input. Ακόμα η SimpleGame μας παρέχει φως. Ένα φως με κατεύθυνση που βρίσκεται στην θέση (100,100,100). Το οποίο είναι προσκολλημένο σε ένα LightState που ονομάζεται και lightState και είναι προσκολλημένο στο rootNode. Ο φωτισμός αυτός θα επηρεάσει κάθε στοιχείο που έχει προστεθεί στην σκηνή εκτός και αν πατήσουμε το πηκτό L η καλέσουμε την lightState.setEnabled(false) όπου απενεργοποιείται.

Επιπλέον η SimpleGame δημιουργεί έναν timer για να παρακολουθεί την αναλογία των frame (frame rate) και τον χρόνο μεταξύ των frames. Ο χρόνος μεταξύ των frames tpf (time per frames) περνάει στις μεθόδους update και render και ειδοποιεί το σύστημα για την ταχύτητα του υπολογιστή. Όταν χρησιμοποιούμε την SimpleGame πάντα θα έχουμε στο κάτω μέρος της οθόνης μας ενημέρωση για το frame rate καθώς και για τα τρίγωνα και τις κορυφές που εμφανίζονται. Αυτά ελέγχονται μέσω του fpsNode και του fpsText όπου το δεύτερο ανανεώνεται αυτόματα από την μέθοδο update της SimpleGame.

Ίσως όμως το πιο σημαντικό αντικείμενο που δημιουργεί η SimpleGame είναι το rootNode. Το rootNode είναι ένας κόμβος χώρου

(spatial Node) όπου ορίζει το αρχικό επίπεδο (την ρίζα) της σκηνής που θα πρέπει να σχεδιαστεί. Προσθέτοντας και αλλά Spatial στο rootNode h SimpleGame θα χειριστεί το rendering και το update τους.

Η SimpleGame παρέχει μεθόδους που μπορούν να γίνουν override και να έχουμε καλύτερο έλεγχο της εφαρμογής μας. Αυτές οι μέθοδοι είναι:

simpleUpdate Παρέχει ένα τρόπο για να προσθέτουμε και άλλες λειτουργίες στην φάση αναβάθμισης του βρόχου. Για παράδειγμα αν περιστρέψουμε ένα αντικείμενο η αν ανιχνευτεί σύγκρουση μεταξύ αντικειμένων αυτά καλούνται μετά την update αλλά πριν από γίνει ανανέωση των γεωμετρικών δεδομένων της σκηνής.

simpleRender. Παρέχει τρόπους για να γίνεται αναπαράσταση δεδομένων τα οποία δεν περιέχονται στο δεδομένο rootNode. Για παράδειγμα αν κάνουμε rendering σε ένα texture αυτές τις κλήσης θα τις τοποθετήσουμε στην simpleRender. Καλούνται μετά την render αλλά πριν γίνει το rendering των στατιστικών ώστε να διασφαλίσουμε ακρίβεια στην αναπαράσταση.

DisplaySystem. Έχει δύο κυρίως δουλειές να δημιουργήσει ένα window και τον Renderer. Και τα δυο δημιουργούνται χρησιμοποιώντας την μέθοδο createWindow. Η μέθοδος αυτή παίρνει το μήκος και το πλάτος του παραθύρου (ανάλυση), και συχνότητα της οθόνης και την επιλογή για fullsreen η όχι. Αυτές οι παράμετροι χρησιμοποιούνται για την δημιουργία του παραθύρου και έπειτα την δημιουργία του Renderer που μας παρέχει η OpenGL. Σε αυτό το σημείο το σύστημα μας είναι έτοιμο για Rendering.Η εφαρμογή του SimpleGame δημιουργεί από μόνη της το DisplaySystem και εμείς δεν χρειάζεται να κάνουμε τίποτα.

Renderer

Ο Renderer έχει την δυνατότητα να πάρει ένα screenshot με ότι υπάρχει εκείνη την στιγμή στην οθόνη. Αυτό ουσιαστικά έχει ως προϋπόθεση να ληφθούν τα δεδομένα από το τελευταίο frame που έχει γίνει render και να αποθηκευτούν σαν μια εικόνα png. Μια κλήση στη μέθοδο takeScreenShot είναι αρκετή ώστε να δημιουργηθεί το αρχείο png αποθηκεύοντας το στο τρέχον directory

Texture Renderer

Ο Texture Renderer παρέχει τα μέσα για να αποδοθεί (render) μια εξ' ολοκλήρου σκηνή σε κάλυμμα (texture) από ότι η σκηνή από μόνη της. Αυτό το Texture τώρα μπορεί να εφαρμοστεί πάνω σε οποιαδήποτε αντικείμενο όπως ένα οποιοδήποτε texture. Η χρησιμότητα του TextureRenderer είναι πανομοιότηπη με του Renderer. Υποστηρίζει την δική του κάμερα επιτρέποντας το σημείο οπτικής να είναι διαφορετικό από αυτό της κύριας σκηνής. Όπως στο Renderer η δημιουργία του TextureRenderer γίνετε μέσω της κλάσης DisplaySystem. Οι σκηνές τότε μπορούν να αποδοθούν σε ένα προμηθευόμενο Texture με μια κλήση στη μέθοδο render

RenderQueue

Το RenderQueue παρέχει μεθόδους για την οργάνωση των στοιχείων της σκηνής πριν την απεικόνιση τους. Το RenderQueue δεν είναι απαραίτητο και μπορεί να παραληφθεί. Παρ' όλα αυτά αργά η γρήγορα θα χρειαστεί να γίνει κάποια γεωμετρική ταξινόμηση για να αποφευχθούν τυχόν λάθη αναπαράστασης. Για παράδειγμα εξ΄ορισμού στην γεωμετρία γίνεται render με βάση τη θέση του αντικειμένου στο scenegraph. Αν υπάρχουν διαυγή-διάφανα αντικείμενα στην σκηνή και απεικονιστούν πριν τα αδιαφανή αντικείμενα που υποτίθεται ότι είναι από πίσω τους τότε αυτά τα αντικείμενα δεν θα είναι ορατά μέσα από τα διαφανή αντικείμενα. Εδώ είναι που χρειάζεται και η χρήση του RenderQueue.

Το RenderQueue χρησιμοποιεί έναν αριθμό από διαφορετικούς "κάδους" για να κάνει ταξινόμηση των αντικειμένων με βάση της παρακάτω

ιδιότητες: αδιαφανή, διαφανή και ortho. Ο Renderer προσδιορίζει της σταθερές και για να ορίσουμε την σειρά (Queue) χρησιμοποιούμε την μέθοδο setRenderQueueMode. Π.χ. για να ορίσουμε ένα αντικείμενο obj σαν αδιαφανή θα γράφαμε:

obj.setRenderQueueMode(Renderer.QUEUE_TRANSPARENT);

Αφού μπουν τα αντικείμενα σε μια σειρά, μπαίνουν σε σειρά προτεραιότητας πριν απεικονιστούν. Η σειρά είναι η ακόλουθη :

1. Απεικονίζονται τα αδιαφανή αντικείμενα (από μπροστά προς τα πίσω)
2. Απεικονίζονται τα διαφανή αντικείμενα από πίσω προς τα μπροστά ζωγραφίζοντας το εσωτερικό του αντικειμένου
3. Απεικονίζοντας τα διαφανή αντικείμενα (από πίσω προς τα μπροστά) σχεδιάζοντας το εξωτερικό των αντικειμένων
4. Απεικόνιση αντικειμένων ortho

Η σειρά που ακολουθείται είναι σημαντική για τους παρακάτω λόγους :

- Απεικονίζοντας πρώτα τα αδιαφανή αντικείμενα διασφαλίζουμε ότι όλα τα αδιαφανή αντικείμενα θα είναι ορατά πίσω από διαφανή αντικείμενα. Απεικονίζοντας από πίσω προς τα μπροστά επιτρέπει στην OpenGL να βελτιστοποιήσει την απεικόνιση παραλείποντας pixels που είναι “υπερφορτωμένα” (occluded)
- Απεικονίζοντας τα διαφανή αντικείμενα από πίσω προς τα μπροστά διασφαλίζει ότι όλα τα αντικείμενα θα είναι ορατά πίσω από ένα διάφανο αντικείμενο
- Απεικονίζοντας τα διαφανή αντικείμενα 2 φορές (μια αποδίδοντας το εσωτερικό και μια το εξωτερικό μέρος) επιτρέπει σε περίπλοκα αντικείμενα να φαίνονται σαν να είναι πραγματικά διάφανα.
- Τα ortho αντικείμενα σχεδιάζονται τελευταία γιατί είναι αντικείμενα που τοποθετούνται στην οθόνη (π.χ. HUD) και όχι στην σκηνή και πρέπει πάντα να βρίσκονται μπροστά από κάθε αντικείμενο σκηνής

ΒΑΣΙΚΑ ΣΧΗΜΑΤΑ

Το JME παρέχει μέσα για την δημιουργία βασικών σχημάτων. Έτσι μας επιτρέπει να βάλουμε πολύ εύκολα και γρήγορα 3D σχήματα στην σκηνή μας. Κάποια από τα βασικά σχήματα είναι:

Box

Υπάρχουν 2 τρόποι για να δημιουργήσουμε ένα κουτί

Α) με min και max points

Vector3f min = new Vector3f (-5,-5,-5);

Vector3f max == new Vector3f (5,5,5);

Box b = new Box("box",min,max);

Β) ορίζοντας κέντρο και πλευρές

Box c = new Box("box" , new Vextor3f(-10,-10,-10),1000,1,1000);

Όπου Vextor3f(-10,-10,-10) είναι η γωνία του κουτιού και τα επόμενα τρία ορίσματα που ακολουθούν είναι το μήκος, το ύψος και το πλάτος αντίστοιχα.

Cylimder

Ένας άπειρος κύλινδρος ορίζεται ως ένα σύνολο σημείων και μία σταθερή απόσταση από μία γραμμή. Ένας πεπερασμένος κύλινδρος είναι ένα υποσύνολο του άπειρου κυλίνδρου όπου το μήκος του είναι λιγότερο από ένα ορισμένο σημείο. Στο JME ο κύλινδρος είναι πάντα ένας πεπερασμένος κύλινδρος. Η δημιουργία ενός κυλίνδρου απαιτεί το ύψος και την ακτίνα. Ο αριθμός των τμημάτων που αποτελούν το ακτινωτό των κυλίνδρου καθώς και το ύψος παρέχονται από μας. Όσο μεγαλύτερος ο αριθμός των τμημάτων τόσο πιο λεπτομερής θα είναι και ο κύλινδρος. Ο κύλινδρος προσανατολίζεται πάντα ξαπλώνοντας κατά μήκος του άξονα Z.

Δημιουργία cylinder:

Cylinder c = new Cylinder(“Cyl” ,10,10,5,20);

10 είναι τα τμήματα για το ύψος και το ακτινωτό.
5 είναι η ακτίνα άρα ο κύλινδρος θα είναι διαμέτρου 10.
20 είναι το ύψος.

Sphere

Μια σφαίρα καθορίζεται από ένα σύνολο σημείων που ισαπέχουν από ένα κεντρικό σημείο. Στο jme η σφαίρα καθορίζεται από ένα κεντρικό σημείο, μια ακτίνα και έναν αριθμό δειγμάτων για τον άξονα Z και τον ακτινωτό άξονα. Οσο μεγαλύτερος ο αριθμός δειγμάτων (ή αριθμός τριγώνων) τόσο καλύτερη η ανάλυση της σφαίρας,

Δημιουργία σφαίρας
Sphere s = new Sphere(“Sphere”,63,50,25);

Ο Ρόλος των Μαθηματικών στο jME

Τα μαθηματικά παίζουν σημαντικό ρόλο και στο jme γιατί αναμφισβήτητα τα 3D γραφικά απαιτούν αρκετές γνώσεις μαθηματικών. Ωστόσο το jme έχει την δυνατότητα και απαλλάσσει τον χρήστη από πολλές λεπτομέρειες καλύπτοντας ένα μεγάλο μέρος των μαθηματικών “υψηλού επιπέδου” Θεμελιώδεις τύποι αναλαμβάνουν την αντιπροσώπευση σύνθετων μαθηματικών εννοιών και την λειτουργία τους χωρίς να χρειάζεται από μας κάποια ιδιαίτερη εμβάθυνση.

Τα διανύσματα αποτελούν στοιχειώδες τύπο σε 3D περιβάλλον και χρησιμοποιούνται κατά κόρον. Οι πίνακες είναι επίσης ένα βασικό στοιχείο για την απεικόνιση γραμμικών συστημάτων. Το Quaternion είναι ίσως το πιο περίπλοκο από τους βασικούς τύπους και στο jme χρησιμοποιείται για την περιστροφή αντικειμένων.

Τα Vectors (διανύσματα) χρησιμοποιούνται για να αντιπροσωπεύσουν κάποιο σημείο ή διεύθυνση ενώ οι πίνακες χρησιμοποιούνται συνήθως για

περιστροφές σε συνδυασμό με το quaternion . Η κλάση Vector3f είναι ίσως η πιο πολυχρησιμοποιημένη κλάση στο jme

Quaternions

Το jme χρησιμοποιεί τα quaternions επειδή επιτρέπουν τις συμπαγείς αντιπροσωπεύσεις των περιστροφών ή αντίστοιχα τους προσανατολισμούς στον τρισδιάστατο χώρο. Με τέσσερις μόνο float τιμές μπορούμε να αναπαραστήσουμε τον προσανατολισμό ενός αντικειμένου όταν ένας πίνακας περιστροφής θα απαιτούσε 9. Ενώ είναι αρκετά δύσκολο να κατανοηθεί πλήρως η έννοια του quaternion υπάρχει ένας μεγάλος αριθμός πιστικών μεθόδων που μας επιπρέπει να τις χρησιμοποιούμε χωρίς να είναι απαραίτητο να κατανοούμε όλα τα μαθηματικά που κρύβονται από πίσω. Βασικά αυτές οι μέθοδοι δεν περιλαμβάνουν τίποτα περισσότερο από το να θέτουν τις τιμές των quaternion x,y,z,w χρησιμοποιώντας άλλα μέσα για της περιστροφές.

Angle Axis

Μπορούμε εάν θέλουμε να ορίσουμε την περιστροφή μας ως προς ένα ζευγάρι άξονα γωνίας (Angle Axis) δηλαδή καθορίζουμε έναν άξονα περιστροφής και την γωνία περιστροφής γύρο από αυτόν τον άξονα. Το quaternion ορίζει μια μέθοδο την fromAngleAxis και την fromAngleNormalAxis άστε να δημιουργήσει ένα quaternion από αυτό το ζευγάρι. Μπορούμε επίσης να προκαλέσουμε περιστροφή από ένα ήδη υπάρχων quaternion χρησιμοποιώντας την μέθοδο toAngleAxis.

Παράδειγμα περιστροφής χρησιμοποιώντας την fromAngleAxis:
περιστροφή στον άξονα Y κατά 1π

```
Vector3f axis = new Vector3f(0,1,0);
Float angle = 3,14f;
s.getLocalRotation().fromAngleAxis(angle,axis);
```

Three Angles

Μπορούμε επίσης να περιστρέψουμε ένα αντικείμενο καθορίζοντας τρεις γωνίες. Οι γωνίες αντιπροσωπεύουν την περιστροφή του κάθε άξονα. Παίρνοντας σαν όρισμα έναν πίνακα τριών στοιχείων float ορίζουν την γωνία, όπου το πρώτο στοιχείο είναι το x, το δεύτερο το y και το τρίτο το z. Η μέθοδος που παρέχεται από το quaternion είναι η fromAnlges και μπορούμε επίσης να γεμίσουμε έναν πίνακα χρησιμοποιώντας την toAngles.

Παράδειγμα περιστροφής χρησιμοποιώντας την fromAngles:
περιστροφή 1 rad στον x, 3 στον y και 0 στον z

```
float[] angles = {1, 3, 0};
s.getLocalRotation().fromAngles(angles);
```

Three Axes

Εάν έχουμε 3 άξονες για να καθορίζουν την περιστροφή μας τότε οι άξονες θα καθορίζουν τον αριστερό άξονα, τον επάνω άξονα και τον κατευθυντικό άξονα αντίστοιχα. Σε αυτήν την περίπτωση μπορούμε να χρησιμοποιήσουμε την fromAxes για να δημιουργήσουμε το quaternion. Πρέπει να σημειώσουμε ότι σε αυτήν την περίπτωση θα δημιουργηθεί ένας καινούργιος πίνακας όπου μετά θα είναι άχρηστος, γι΄αυτό και αυτή η μέθοδος δεν είναι καλό να χρησιμοποιείται εάν είναι να κληθεί πολλές φορές.

Visibility Determination

Ο προσδιορισμός την εμβέλειας των ορατών στοιχείων ασχολείται με την ελαχιστοποίηση των δεδομένων των στοιχείων που στέλνονται στην κάρτα γραφικών για να αποδοθούν στην οθόνη. Συγκεκριμένα δεν θέλουμε να στέλνουμε στοιχεία και δεδομένα τα οποία δεν πρόκειται να εμφανιστούν. Τα στοιχεία τα οποία δεν στέλνονται στην κάρτα γραφικών χαρακτηρίζονται ως culled. Η αρχική εστίαση αυτού του τμήματος είναι το frustrum Culling που βασίζεται στο οπτικό frustrum της κάμερας.

Ουσιαστικά αυτό το frustrum δημιουργεί έξιi τυποποιημένα επίπεδα οπτικής. Εξετάζονται τα Bounding Volumes των αντικειμένων για να ελεγχθεί εάν τα όρια τους αυτά βρίσκονται εντός εμβέλειας του frustrum. Εάν σε οποιοδήποτε σημείο η οριοθέτηση του αντικειμένου είναι έξω από το επίπεδο που ορίσαμε, πετιέται έξω και δεν επεξεργάζονται πλέον ώστε να αποδοθούν στην σκηνή. Αυτό περιλαμβάνει και τυχόν “παιδία” τα οποία διαχειρίζονται αυτό επιτρέπει γρήγορα culling για μεγάλα τμήματα σκηνής.

Collision Calculations

Το jme παρέχει ένα εκτενές σύστημα ανίχνευσης συγκρούσεων. Χρησιμοποιώντας το Bounding Volumes μπορούν να γίνει ελεγχος για το εάν δυο αντικείμενα συγκρούονται η όχι.

ΚΕΦΑΛΑΙΟ 3.

Camera

Το Jme χρησιμοποιεί την κάμερα για να περιγράψει διάφορους βασικούς όρους σχετικά με το πώς γίνεται render στο scenegraph. Όπως και σε μια πραγματική φωτογραφική μηχανή η θέση και ο φακός καθορίζουν πόσο και ποιο μέρος του πραγματικού κόσμου θα καταγραφεί για να αποδοθεί αργότερα. Έτσι και εδώ η κάμερα του jme καθορίζει πόσος από τον εικονικό μας κόσμο θα αποδοθεί και θα επιδειχθεί στο χρήστη.

Την κάμερα μπορούμε να την χειριστούμε όπως οποιοδήποτε άλλο αντικείμενο πάνω στην σκηνή. Αυτό απλά απαιτεί να κάνουμε χρήση του CameraNode. Η τοποθεσία και οι ρυθμίσεις τις κάμερας μας καθορίζουν το λεγόμενο View Frustrum το οποίο είναι μια λογική περιγραφή του συνολικού όγκου του χώρου που είναι ορατός στον χρηστή.

Εάν παίρναμε μια πυραμίδα, κόβαμε το πάνω μέρος της και κοιτάζαμε την πυραμίδα από την κορυφή προς την βάση αυτό που βλέπουμε είναι αυτό που ορίζεται ως view frustrum. Η νέα διαμόρφωση κορυφής η οποία

δημιουργείται αντιπροσωπεύει την οθόνη του χρήστη. Η πυραμίδα επεκτείνεται προς τα έξω διευρύνοντας το τοπικό πεδίο. Οτιδήποτε βρίσκεται εντός της πυραμίδας γίνεται render ενώ οτιδήποτε βρίσκεται εκτός αγνοείται.

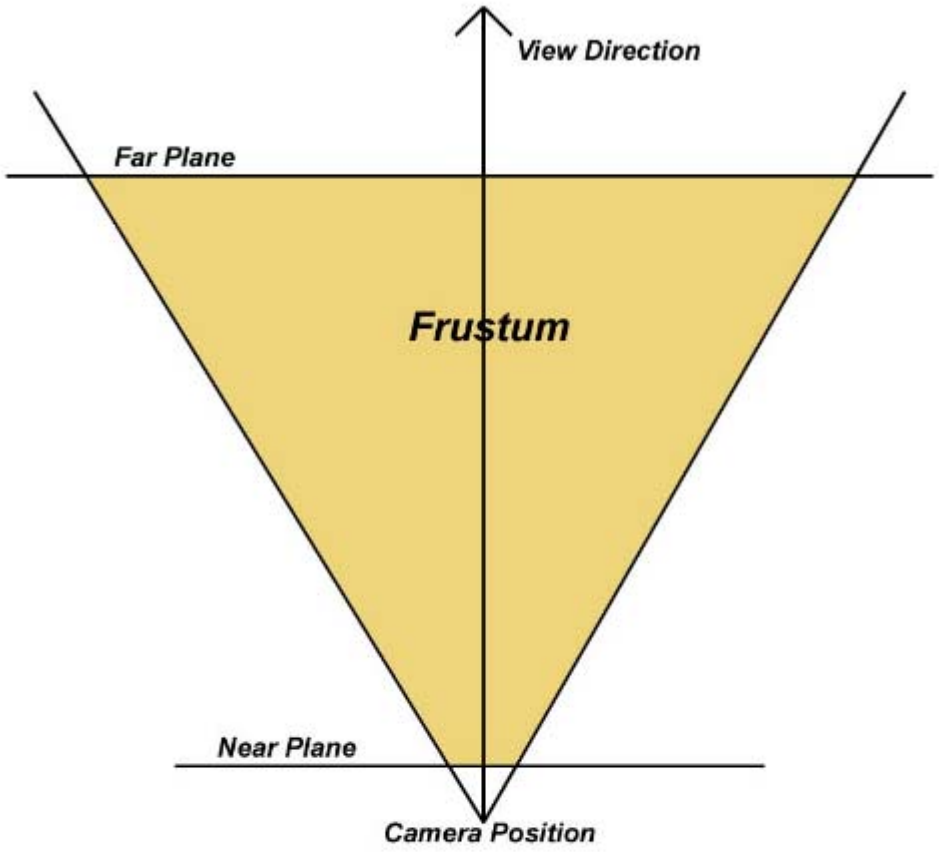


Figure 1: 2D View Frustum

Αυτό δίνει την δυνατότητα στο Jme να χειρίζεται σύνθετα scenegraph με πολλά αντικείμενα διατηρώντας ωστόσο υψηλό Frame Per Second (fps). Οι ρυθμίσεις της κάμερας μας θα καθορίσουν ποιο θα είναι το view Frustrum, ο προσανατολισμός της στον κόσμο και ποιες οι ιδιότητες των πλευρών της π.χ. οι πλευρές εκτείνονται με τέτοιο τρόπο που να δημιουργούν μια μεγάλη γωνία δίνοντας στον χρήστη ένα πολύ ευρύ οπτικό πεδίο ή μένουν σχετικά κοντά δημιουργώντας την αίσθηση ενός τούνελ στην σκηνή;

Επιπλέον η κάμερα επιτρέπει στο χρήστη να ελέγχει ποιο είναι το frustrum και ποιος ο προσανατολισμός του. Στο jme δεν θα πρέπει ποτέ να δημιουργούμε άμεσα μια κάμερα έτσι ώστε να αποφύγουμε τυχόν εξαρτήσεις

από κάποιο συγκεκριμένο API. Εναλλακτικά ζητάμε μια κάμερα από τον Renderer. Η μόνη παράμετρος που απαιτείται είναι η ανάλυση που θα αποδοθεί η εικόνας μας ώστε να έχουμε ακριβή απόδοση.

Το view Frustrum

Στο jme το frustrum μέσα στην κλάση της κάμερας αποτελείται από έξι επίπεδα (top, bottom, front,back,left,right) Υπάρχουν δυο τρόποι για να ορίσουμε το View Frustrum.

1.

public void setFrustum(float near, float far, float left, float right, float top, float bottom);

χρησιμοποιώντας την setFrustrum ο χρήστης καθορίζει τα έξη επίπεδα οπτικής του frustrum. Αυτά τα επίπεδα ορίζονται ως η απόσταση από το σημείο του ματιού.

Για παράδειγμα το

cam.setFrustum(1.0f, 1000.0f, -0.55f, 0.55f, 0.4125f, -0.4125f);

ορίζει το κοντινό επίπεδο ως μια μονάδα από το σημείο του ματιού, το μακρινό επίπεδο ως 1000 μονάδες, το left και right ως 0.55 (κάνοντας έτσι το κοντινό επίπεδο να έχει πλάτος 1,1) και το top και botton από 0,4125 (κάνοντας το κοντινό επίπεδο ύψους 0,85)

2.

public void setFrustumPerspective(float fovY, float aspect, float near, float far);

όπου το foy(Y) ορίζει την γωνία οπτικής στην κατεύθυνση Y. Το aspect καθορίζει την σχέση μεταξύ του ύψους και του πλάτους του παραθύρου και το near και far καθορίζουν το κοντινό και το μακρινό επίπεδο.

Για παράδειγμα
cam.setFrustumPerspective(45.0f,(float) display.getWidth() / (float) display.getHeight(), 1, 1000);

Scenegraph

Μια αναπαράσταση σκηνής (scenegraph) αποτελείται από δύο τύπους κόμβων. Τους εσωτερικούς (Internal Nodes) και τους φυλλώδες (Leaf Nodes). Οι εσωτερικοί κομβόι αναφέρονται και ως Nodes και οι LeafNodes αναφέρονται ως Geormetry (γεωμετρίες). Η διαφορά των δύο αυτών είναι ότι τα Nodes μπορούν να περιέχουν παιδία (π.χ. άλλο κόμβο ή άλλη γεωμετρία) ενώ οι γεωμετρίες δεν μπορούν να έχουν παιδιά, ενώ και οι δυο τύποι περιέχουν σημαντικές πληροφορίες για τον εαυτό τους (transforms,Boounding Volumes, renderState και controllers).

Spatials

Το Spatial καθορίζει την βασική κλάση για όλα τα στοιχεία στο scenegraph. Έχουμε δύο κατηγορίες Spatial τα Geometry και Nodes. Οι πληροφορίες του scenegraph δημιουργούνται σε 2 φάσεις – περάσματα. Στο πρώτο πέρασμα που γίνεται από πάνω προς τα κάτω γίνονται όλες οι μετατροπές (transforms) και στο δεύτερο πέρασμα πουυ γίνεται από κάτω προς τα πάνω γίνονται οι οριοθετήσεις.

CameraNode

Η CameraNode μας επιτρέπει να χρησιμοποιήσουμε ο αντικείμενο Camera σαν στοιχείο του scenegaph. Το όφελος από αυτό είναι ότι μπορούμε να προσκολλήσουμε την Camera μας σε οποιοδήποτε αντικείμενος την σκηνής. Ο προσανατολισμός της CameraNode καθορίζει και τον προσανατολισμό της camera. Κατά την κλήση της UpdateWorldData η CameraNode θα εξασφαλίσει την ανανέωση του οπτικού πεδίου της κάμερας. Κατά συνέπεια αν καλέσουμε την updateGeometricState από την σκηνή

πάνω στην οποία είναι προσκολλημένη η CameraNode θα ανανεωθεί το οπτικό πεδίο.

ΚΕΦΑΛΑΙΟ 4.

Model Bound

Τα όρια ενός μοντέλου καθορίζονται από το Boundong Volume που περιέχει τα γεωμετρικά δεδομένα προέλευσης του. Τα όρια αυτά δεν μεταφράζονται σε χώρος τον κόσμο μας αλλά κρατιούνται στο χώρο γεωμετρίας. Αυτά τα όρια χρησιμοποιούνται για να μετρήσουμε τα όρια των gometry και κατ' επέκταση τα όρια των κόμβων.

Bounding Volumes

Το Bounding Volumes χρησιμοποιείται για την αφαίρεση περιττών αντικειμένων πριν το render της σκηνής και καθορίζει εάν ένα αντικείμενο βρίσκεται μέσα στο οπτικό πεδίο μιας κοινότητας. Για σύνθετα αντικείμενα όπως ένας ανθρώπινος οργανισμός είναι πολύ δύσκολο να καθοριστεί υπολογίστηκα εάν αυτό το αντικείμενο βρίσκεται μέσα στο οπτικό που μας ενδιαφέρει δεδομένου ότι όλες οι δοκιμές θα έπρεπε να γίνουν για κάθε πολύγωνο που υπάρχει στο μοντέλο.

Έτσι ένας αόρατος απλός μαθηματικός όγκος όπως μια σφαίρα ή ένας κύβος τοποθετείται γύρο από το πρότυπο με τέτοιο τρόπο ώστε να περιλαμβάνει όλη την γεωμετρία του σύνθετου προτύπου. Όταν λείπουν γίνεται η διαδικασία του culling στο scenegraph αυτό το απλό αντικείμενο εξετάζεται για το αν θα είναι ορατό ή όχι πριν σταλεί για rendering. Εάν το απλό αυτό αντικείμενο αποτύχει το τεστ (δεν βρίσκεται μέσα στο οπτικό πεδίο της κάμερας) τότε το σύνθετο μοντέλο και όλα του τα παιδιά απορρίπτονται από την διαδικασία του render κερδίζοντας έτσι δύναμη στον επεξεργαστή

CullState

Καθορίζει ένα RenderState[1] που χειρίζεται την πρόσοψη ενός συγκεκριμένου τριγώνου. Υπάρχουν τρεις επιλογές : CS_FRONT, CS_BACK, CS_NONE. Η πιο συνηθισμένη χρήση του CullState είναι η CS_BACK (αφαίρεση του πίσω μέρους των αντικειμένων. Με αυτόν τον τρόπο επιτρέπει στον Renderer να μην υπολογίζει τα τρίγωνα τα οποία δεν βλέπουν προς την κάμερα αυτό μπορεί να έχει σαν αποτέλεσμα σημαντική αύξηση της ταχύτητας. Το CullState χειρίζεται όπως οποιαδήποτε άλλο RenderSate και μπορεί να τοποθετηθεί οπουδήποτε στην σκηνή μας. Εξ' ορισμού καμία πλευρά δεν απορρίπτεται

LightState

Παρέχει τα μέσα ώστε να μπορεί να προστεθεί φωτισμός σε μια γραφική σκηνή. Ενα lightState μπορεί να περιέχει καθόλου ή πολλά φώτα. Το LigthState μπορούμε να το χειριστούμε καλύτερα χρησιμοποιώντας το μέσο ενός LightNode. Με το LightNode μπορούμε αν χειριστούμε το φως σαν να ήταν αντικείμενο της σκηνής.

ZBufferState (ή depthBuffer)

Το ZBufferState βοηθάει στο rendering. Αυτό που κάνει είναι να κρατάει πληροφορίες για το κάθε pixel π.χ. το χρώμα, το πόσο μακριά βρίσκεται αυτό το pixel καθώς και αν μπορεί να το κάνει override ή όχι. Το τελευταίο εξαρτάται από το τι ορίσματα θα του δώσουμε εμείς.

- CF_NEVER - το πρώτο pixel που τοποθετείται να μην καλυφθεί από κανένα επόμενο.

[1] Ενώ η γεωμετρία καθορίζει τα δεδομένα που πρόκειται να γίνουν render το RenderState καθορίζει την εμφάνιση των δεδομένων.

- CF_LESS – έχουμε αντικατάσταση pixel αν το καινούργιο pixel βρίσκεται πιο κοντά (σε βάθος) από τα τρέχον.
- CF_EQUAL – γίνεται αντικατάσταση pixel αν το καινούργιο pixel βρίσκεται στο ίδιο βάθος με το ήδη υπάρχον.
- CF_LEQUAL - γίνεται αντικατάσταση pixel αν το καινούργιο pixel βρίσκεται σε μικρότερο ή ίσο βάθος από το ήδη υπάρχον.
- CF_GREATER - γίνεται αντικατάσταση pixel αν το καινούργιο pixel βρίσκεται πιο μακριά από το τρέχον pixel.
- CF_NOTEQUAL - γίνεται αντικατάσταση pixel αν το καινούργιο pixel δεν βρίσκεται στο ίδιο βάθος από το ίση υπάρχον.
- CF_GEQUAL - γίνεται αντικατάσταση pixel αν το καινούργιο pixel βρίσκεται σε ίσο η μεγαλύτερο βάθος από το ήδη υπάρχον..
- CF_ALWAYS – πάντα το καινούργιο pixel αντικαθιστά τα ήδη υπάρχον.

ΚΕΦΑΛΑΙΟ 5.

jME Physics

Το σύστημα του jME physics παρέχει ένα interface ανάμεσα στο jME και στην ODE (Open Dynamic Engine). Τοποθετείται πάνω από μια ελαφρός τροποποιημένη έκδοση της odejava και παρέχει τρόπους για να δημιουργηθεί πολύ εύκολα ένας κόσμος φυσικής και να προστεθούν σε αυτόν αντικείμενα. Μπορούμε να έχουμε κάτι πολύ απλό όπως ένα κουτί να εκτελεί μια ελεύθερη πτώση έως κάτι πολύ σύνθετο π.χ. ένα αυτοκίνητο ή έναν άνθρωπο. Η αρχική ιδέα ήταν του Nicolaas de Bruyn που έβαλε τις βάσεις ενώ αργότερα προστέθηκαν και οι Per Thulin και Ahmed Al-Hindawi σε μια προσπάθεια για ανάπτυξη και πρόσθεση νέων χαρακτηριστικών.

ODE (Open Dynamic Engine)

Είναι μια open source βιβλιοθήκη φυσικής γραμμένη σε C. Με άλλα λόγια είναι μια ελεύθερη βιβλιοθήκη για μιμήσεις αυστηρά δυναμικής

σωμάτων. Για παράδειγμα επίγειων οχημάτων, πλασμάτων με πόδια καθώς και κίνηση αντικειμένων σε εικονικές πραγματικότητες. Η ODE είναι γρήγορη, εύκαμπτη, ανεξάρτητη πλατφόρμας με προηγμένες ενώσεις (joints), επαφές με τριβή, ανίχνευση συγκρούσεων κ.α. Εν ολίγοις είναι μια βιβλιοθήκη κατάλληλη για φυσικές προσομοιώσεις.

ODEjava

Είναι μια open dynamic Engine αποκλειστικά για java. Το API παρουσιάζει πρόσβαση, και χαμηλού επιπέδου στην βιβλιοθήκη της ODE και πρόσβαση υψηλού επιπέδου στα αντικείμενα της odejava. Καθώς η βιβλιοθήκη είναι γραμμένη σε C δεν είναι αντικειμενοστραφής και γι' αυτό το λόγο χρειάστηκε να δημιουργηθούν τα αντικείμενα υψηλότερου επιπέδου. Η odejava μπορεί να χρησιμοποιηθεί σε οποιαδήποτε εφαρμογή γιατί είναι μια ανεξάρτητη βιβλιοθήκη. Η πιο κοινή χρήση της odejava είναι σε 3D εφαρμογές.

ΚΕΦΑΛΑΙΟ 6.

jME Στην Πράξη

Η παρακάτω εφαρμογή δημιουργεί έναν κόσμο φυσικής. Στον κόσμο αυτό απεικονίζεται μια πίστα και ένα αντικείμενο το οποίο κατευθύνουμε εμείς με τα πλήκτρα W,A,S,D, καθως και μετά Q, E που μετακινούν την πίστα. Υπάρχει επίσης μια ChaseCamera η οποία ακολουθεί πιστά το αντικείμενο μας σε κάθε του κίνηση. Το αντικείμενο μας έχει οριστεί ως dynamicNode ενώ το terrain και τα διαχωριστικά ως staticNode, έτσι μπορούμε να έχουμε collision.

```
/*
*
* jME sthn praksi
* Anaptuksh Trisdiastatou diadrasthkoy pexnidioy perihghshs
*
*/
```

```
package simplesimple;

import com.jme.app.SimpleGame;
import com.jme.bounding.BoundingBox;
import com.jme.bounding.BoundingSphere;
import com.jme.image.Texture;
import com.jme.input.KeyBindingManager;
import com.jme.input.KeyInput;
import com.jme.input.NodeHandler;
import com.jme.intersection.Intersection;
import com.jme.math.Vector3f;
import com.jme.math.FastMath;
import com.jme.renderer.Camera;
import com.jme.scene.CameraNode;
import com.jme.scene.Controller;
import com.jme.scene.Node;
import com.jme.scene.Skybox;
import com.jme.scene.Text;
import com.jme.scene.TriMesh;
import com.jme.scene.shape.Box;
import com.jme.scene.state.AlphaState;
import com.jme.scene.state.RenderState;
import com.jme.scene.state.TextureState;
import com.jme.system.DisplaySystem;
import com.jme.util.LoggingSystem;
import com.jme.util.TextureManager;
import com.jmex.effects.ParticleManager;
import com.jmex.model.XMLparser.Converters.FormatConverter;
import com.jmex.model.XMLparser.Converters.MaxToJme;
import com.jmex.model.XMLparser.JmeBinaryReader;
import com.jmex.physics.DynamicPhysicsNode;
import com.jmex.physics.StaticPhysicsNode;
import com.jmex.physics.geometry.PhysicsBox;
import com.jmex.physics.util.SimplePhysicsGame;
import java.io.BufferedInputStream;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.net.URL;
import java.text.NumberFormat;
import java.util.Calendar;
import java.util.HashMap;
import java.util.logging.Level;
import com.jme.scene.shape.Box;
import com.jme.math.Vector3f;
import com.jmex.physics.*;
import com.jme.math.Quaternion;
import com.jme.light.PointLight;
import com.jme.scene.state.LightState;
```

```
import com.jme.renderer.ColorRGBA;
import com.jme.scene.SharedMesh;
import com.jme.light.DirectionalLight;
import com.jme.bounding.BoundingBox;
import com.jme.scene.shape.Cylinder;
import com.jme.input.InputHandler;
import com.jme.input.InputHandler;
import com.jme.input.action.InputAction;
import com.jme.input.action.InputActionEvent;
import com.jme.input.thirdperson.ThirdPersonMouseLook;
import com.jme.bounding.BoundingSphere;
import com.jme.input.ChaseCamera;
import com.jme.input.ThirdPersonHandler;
import com.jme.scene.shape.Sphere;
import com.jmex.physics.TranslationalJointAxis;
import com.jmex.physics.Joint;
import com.jme.scene.state.ZBufferState;
import com.jme.scene.state.CullState;
import com.jme.scene.Spatial;
import com.jme.math.spring.SpringPoint;
import java.util.Date;
import com.jme.input.util.SyntheticButton;

/**
 *
 * @author Eirini
 */

public class SimpleSimple extends SimplePhysicsGame {

    /** The camera that we see through. */
    private CameraNode camNode;
    private Node cam_char;
    private ChaseCamera chaser;
    /**  Physics Nodes   */
    private StaticPhysicsNode
staticNode,staticNode2,staticNode3,staticNode4,staticNode5;
    private StaticPhysicsNode staticNodeEso3,staticNodeFinish;
    private DynamicPhysicsNode
dynamicNode,dynamicNode2,dynamicNode3,DynamicNodeFinish;
    private DynamicPhysicsNode
DynamicBomba1,DynamicBomba2,DynamicBomba3,DynamicBomba4,DynamicBo
mba5;
    /** the bonus we are going to use */
    private DynamicPhysicsNode
DynamicBonus1,DynamicBonus2,DynamicBonus3,DynamicBonus4;
    boolean reDrawTimer;
    /** parts of the car */
    Box h, topBox, FinishBox;
     Node car;
```

```
    /** the bombes we are going to use */
    Sphere Bonus1,Bonus2,Bonus3,Bonus4;
    /** keeps the date */
    Date  nowDate;
    long end;

    float[] angles;
    /** A sky box for our scene. */
    private Skybox skybox;
    /**  clock */
    private int second =60;
    private int hour;
    private int minute;
    private int startSecond =60;
    private int startMinute =1;
    private int startHour;
    /** variable to check if all bonus are collected */
    private boolean finished;
    /** timer */
    private Calendar event;
    private static final NumberFormat twoDecimals = NumberFormat.getInstance();
    /** texts of the game */
    private static Text
completionText,LivesText,EndLivesText,restartTxt,EndTimeText,BonusText,winTex
t;
    private boolean continueUpdating = true;
    /**variable to keep count of player's lives and bonus */
    private int lives, bonus=0;



    public static void main(String[] a) {
        SimpleSimple app = new SimpleSimple();
        // Signal to show properties dialog
        app.setDialogBehaviour(SimpleGame.ALWAYS_SHOW_PROPS_DIALOG);
        // Start the program
        app.start();
    }




    protected void simpleInitGame() {

        /** Sets the title of our display. */
        display.setTitle("Monkey Box");

        /** Create a ZBuffer to display pixels closest to the camera above farther ones.
*/
        ZBufferState buf = display.getRenderer().createZBufferState();
```

```
    buf.setEnabled(true);
    buf.setFunction(ZBufferState.CF_LEQUAL);
    rootNode.setRenderState(buf);
    // inisialize the variable
    finished = false;
    //add moto to the scene
    setupMoto();
    //add ground to the scene
    setupGround();
    //build the ChaseCamera
    setupChaseCam();
    //build the player input
    setupInput();
    //add lihght to the world
    setupLighting();
    //add external borders to the scene
    setupBorder();
    //add internal borders to the scene
    setupInsideB();
    //Add the Skybox
    setupSkybox();
    //add bonus and bombes to the scene
    setupEmpodia();
    //keep track of lives and bonus
    LivesAndBonus();
    //build the timer for the count down
    Countdown();
    //update the scene graph for rendering
    rootNode.updateGeometricState(0.0F, true);
    rootNode.updateRenderState();


}

private void setupGround() {

    // create object floor
    Box floor = new Box("Floor", new Vector3f(-10,-10,-10), 100, 0.5f, 100);
    //set the new position
    floor.setLocalTranslation(new Vector3f(0,-18,0));
    // Give floor a bounds object to allow it to be culled
    floor.setModelBound(new BoundingBox());
    // Calculate the best bounds for the object
    floor.updateModelBound();
    floor.setLightCombineMode(LightState.OFF);

    //floor texture
    // Get a URL that points to the texture we're going to load
    URL monkeyLoc;
    monkeyLoc=SimpleSimple.class.
```

```
            getClassLoader().getResource(
            "ground.jpg");
        // Get a TextureState
        TextureState ts=display.getRenderer().createTextureState();
        // Use the TextureManager to load a texture
        Texture t = TextureManager.loadTexture(monkeyLoc,
            Texture.MM_LINEAR,
            Texture.FM_LINEAR);  //MM_LINEAR and FM_LINEAR let us know hot
                                    to filter the texture
        // Assign the texture to the TextureState
        ts.setTexture(t);
        // Sets that floor should use renderstate ts
        floor.setRenderState(ts);

        //make floor static since floor can't move'
        // create static Node
        staticNode = getPhysicsSpace().createStaticNode();
        // attach the node to the root node to have it updated each frame
        rootNode.attachChild(staticNode);
        //attach floor to the static node
        staticNode.attachChild(floor);
        //generate the collision geometry
        staticNode.generatePhysicsGeometry();
        // not to be visible the forces on the screen
        showPhysics = false;
        //add transparent floor not to fall the car in space
        StaticPhysicsNode transparentfloor = getPhysicsSpace().createStaticNode();
        rootNode.attachChild(transparentfloor);

        //create a Physics box
        PhysicsBox floorBox = transparentfloor.createBox("floor");
        //scale the floor
        floorBox.getLocalScale().set(300, 0.5F, 300F);
        floorBox.setLocalTranslation(new Vector3f(0,-28,0));
        rootNode.attachChild( transparentfloor);



    }

    private void setupLighting() {

        /** Set up a basic, default light. */
        DirectionalLight light = new DirectionalLight();
        light.setDiffuse(new ColorRGBA(1.0f, 1.0f, 1.0f, 1.0f));
        light.setAmbient(new ColorRGBA(0.5f, 0.5f, 0.5f, 1.0f));
        light.setDirection(new Vector3f(1,-1,0));
        light.setEnabled(true);

        /** Attach the light to a lightState and the lightState to rootNode. */
```

```
    LightState lightState = display.getRenderer().createLightState();
    lightState.setEnabled(false);
    lightState.attach(light);
    rootNode.setRenderState(lightState);
}

private void setupBorder() {

    // create object toixos
    Box toixos1 = new Box("wall1", new Vector3f(0,0,0),10,6,3 );
    // array to keep the angle of rotation
    float [] rot1 ={0,1.f,0};
    //assing a rotation
    toixos1.setLocalRotation(new Quaternion(rot1));
    //set new position
    toixos1.setLocalTranslation(new Vector3f(-80,-21.5f,-85));
    toixos1.setModelBound(new BoundingBox());
    toixos1.updateModelBound();

    //make toixos a static object
    StaticPhysicsNode staticNode2 = getPhysicsSpace().createStaticNode();
    rootNode.attachChild(staticNode2);
    staticNode2.attachChild(toixos1);
    staticNode2.generatePhysicsGeometry();

    //point to the toixos texture
    URL monkeyLoc2;
    monkeyLoc2=SimpleSimple.class.
          getClassLoader().getResource("diagwnia.jpg");

    // Get a TextureState
    TextureState ts2=display.getRenderer().createTextureState();
    // Use the TextureManager to load a texture
    Texture t2 = TextureManager.loadTexture(monkeyLoc2,
          Texture.MM_LINEAR,
          Texture.FM_LINEAR);
    // Assign the texture to the TextureState
    ts2.setTexture(t2);
    // Signal that toixos1 should use renderstate ts
    toixos1.setRenderState(ts2);


    //toixos2

    Box toixos2 = new Box("wall2", new Vector3f(0,0,0),10,6,1 );
    float [] rot2 ={0,0.20f,0};
    toixos2.setLocalRotation(new Quaternion(rot2));
    toixos2.setLocalTranslation(new Vector3f(-30,-21.5f,-107));
    toixos2.setModelBound(new BoundingBox());
    toixos2.updateModelBound();
```

```
toixos2.setRenderState(ts2);
StaticPhysicsNode staticNode3 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNode3);
staticNode3.attachChild(toixos2);
staticNode3.generatePhysicsGeometry();

//toixos3

Box toixos3 = new Box("wall", new Vector3f(0,0,0),10,6,3 );
float [] rot3 ={0,1.57f,0};
toixos3.setLocalRotation(new Quaternion(rot3));
toixos3.setLocalTranslation(new Vector3f(-105,-21.5f,-10));
toixos3.setModelBound(new BoundingBox());
toixos3.updateModelBound();
toixos3.setRenderState(ts2);
StaticPhysicsNode staticNode4 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNode4);
staticNode4.attachChild(toixos3);
staticNode4.generatePhysicsGeometry();

//toixos4

Box toixos4 = new Box("wall", new Vector3f(0,0,0),10,6,3 );
float [] rot4 ={0,1.20f,0};
toixos4.setLocalRotation(new Quaternion(rot4));
toixos4.setLocalTranslation(new Vector3f(-100,-21.5f,-50));
toixos4.setModelBound(new BoundingBox());
toixos4.updateModelBound();
toixos4.setRenderState(ts2);
StaticPhysicsNode staticNode5 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNode5);
staticNode5.attachChild(toixos4);
staticNode5.generatePhysicsGeometry();


//toixos6

Box toixos6 = new Box("wall2", new Vector3f(0,0,0),10,6,1 );
float [] rot6 ={0,0.20f,0};
toixos2.setLocalRotation(new Quaternion(rot2));
toixos6.setLocalTranslation(new Vector3f(15,-21.5f,-110));
toixos6.setModelBound(new BoundingBox());
toixos6.updateModelBound();
toixos6.setRenderState(ts2);
StaticPhysicsNode staticNode7 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNode7);
staticNode7.attachChild(toixos6);
staticNode7.generatePhysicsGeometry();

//toixos7
```

```
Box toixos7 = new Box("wall2", new Vector3f(0,0,0),10,6,3 );
float [] rot7 ={0,2.27f,0};
toixos7.setLocalRotation(new Quaternion(rot7));
toixos7.setLocalTranslation(new Vector3f(60,-21.5f,-90));
toixos7.setModelBound(new BoundingBox());
toixos7.updateModelBound();
toixos7.setRenderState(ts2);
StaticPhysicsNode staticNode8 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNode8);
staticNode8.attachChild(toixos7);
staticNode8.generatePhysicsGeometry();

//toixos8

Box toixos8 = new Box("wall2", new Vector3f(0,0,0),10,6,3 );
float [] rot8 ={0,2.10f,0};
toixos8.setLocalRotation(new Quaternion(rot8));
toixos8.setLocalTranslation(new Vector3f(85,-21.5f,-45));
toixos8.setModelBound(new BoundingBox());
toixos8.updateModelBound();
toixos8.setRenderState(ts2);
StaticPhysicsNode staticNode9 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNode9);
staticNode9.attachChild(toixos8);
staticNode9.generatePhysicsGeometry();

//toixos9

Box toixos9 = new Box("wall2", new Vector3f(0,0,0),10,6,1.5f );
float [] rot9 ={0,1.50f,0};
toixos9.setLocalRotation(new Quaternion(rot9));
toixos9.setLocalTranslation(new Vector3f(89,-21.5f,-10));
toixos9.setModelBound(new BoundingBox());
toixos9.updateModelBound();
toixos9.setRenderState(ts2);
StaticPhysicsNode staticNode10 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNode10);
staticNode10.attachChild(toixos9);
staticNode10.generatePhysicsGeometry();

//toixos10

Box toixos10 = new Box("wall2", new Vector3f(0,0,0),10,6,3 );
float [] rot10 ={0,1.20f,0};
toixos10.setLocalRotation(new Quaternion(rot10));
toixos10.setLocalTranslation(new Vector3f(80,-21.5f,35));
toixos10.setModelBound(new BoundingBox());
toixos10.updateModelBound();
toixos10.setRenderState(ts2);
```

```
StaticPhysicsNode staticNode11 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNode11);
staticNode11.attachChild(toixos10);
staticNode11.generatePhysicsGeometry();

//toixos11

Box toixos11 = new Box("wall2", new Vector3f(0,0,0),10,6,3 );
float [] rot11 ={0,0.80f,0};
toixos11.setLocalRotation(new Quaternion(rot11));
toixos11.setLocalTranslation(new Vector3f(55,-21.5f,65));
toixos11.setModelBound(new BoundingBox());
toixos11.updateModelBound();
toixos11.setRenderState(ts2);
StaticPhysicsNode staticNode12 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNode12);
staticNode12.attachChild(toixos11);
staticNode12.generatePhysicsGeometry();

//toixos12

Box toixos12 = new Box("wall2", new Vector3f(0,0,0),10,6,3 );
float [] rot12 ={0,0.20f,0};
toixos12.setLocalRotation(new Quaternion(rot12));
toixos12.setLocalTranslation(new Vector3f(20,-21.5f,80));
toixos12.setModelBound(new BoundingBox());
toixos12.updateModelBound();
toixos12.setRenderState(ts2);
StaticPhysicsNode staticNode13 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNode13);
staticNode13.attachChild(toixos12);
staticNode13.generatePhysicsGeometry();

//toixos13

Box toixos13 = new Box("wall2", new Vector3f(0,0,0),10,6,3 );
float [] rot13 ={0,0.05f,0};
toixos13.setLocalRotation(new Quaternion(rot13));
toixos13.setLocalTranslation(new Vector3f(-20,-21.5f,83));
toixos13.setModelBound(new BoundingBox());
toixos13.updateModelBound();
toixos13.setRenderState(ts2);
StaticPhysicsNode staticNode14 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNode14);
staticNode14.attachChild(toixos13);
staticNode14.generatePhysicsGeometry();

//toixos14

Box toixos14 = new Box("wall2", new Vector3f(0,0,0),10,6,3 );
```

```
    float [] rot14 ={0,-0.70f,0};
    toixos14.setLocalRotation(new Quaternion(rot14));
    toixos14.setLocalTranslation(new Vector3f(-60,-21.5f,70));
    toixos14.setModelBound(new BoundingBox());
    toixos14.updateModelBound();
    toixos14.setRenderState(ts2);
    StaticPhysicsNode staticNode15 = getPhysicsSpace().createStaticNode();
    rootNode.attachChild(staticNode15);
    staticNode15.attachChild(toixos14);
    staticNode15.generatePhysicsGeometry();

}


private void setupInsideB()   {

    // create object Eso1
    Box eso1 = new Box("wall", new Vector3f(0,0,0),2,3,2 );
    float [] rot1 ={0,2.10f,0};
    eso1.setLocalRotation(new Quaternion(rot1));
    eso1.setLocalTranslation(new Vector3f(20,-24.5f,36));
    eso1.setModelBound(new BoundingBox());
    eso1.updateModelBound();
    StaticPhysicsNode staticNodeEso1= getPhysicsSpace().createStaticNode();
    rootNode.attachChild(staticNodeEso1);
    staticNodeEso1.attachChild(eso1);
    staticNodeEso1.generatePhysicsGeometry();


     //toixos texture
    URL monkeyLoc2;
    monkeyLoc2=SimpleSimple.class.
          getClassLoader().getResource("myborderROT256_512.jpg");
    // Get a TextureState
    TextureState ts2=display.getRenderer().createTextureState();
    // Use the TextureManager to load a texture
    Texture t2 = TextureManager.loadTexture(monkeyLoc2,
          Texture.MM_LINEAR,
          Texture.FM_LINEAR);
    // Assign the texture to the TextureState
    ts2.setTexture(t2);
    eso1.setRenderState(ts2);

    //Eso2

    Box eso2 = new Box("wall", new Vector3f(0,0,0),2,3,2 );
    float [] rot2 ={0,1.10f,0};
    eso2.setLocalRotation(new Quaternion(rot2));
    eso2.setLocalTranslation(new Vector3f(-40,-24.5f,35));
    eso2.setModelBound(new BoundingBox());
```

```
eso2.updateModelBound();
eso2.setRenderState(ts2);
StaticPhysicsNode staticNodeEso2= getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNodeEso2);
staticNodeEso2.attachChild(eso2);
staticNodeEso2.generatePhysicsGeometry();

//eso3

Cylinder eso3= new Cylinder("eso3",20,20,2,10);
Quaternion q = new Quaternion();
q.fromAngleAxis(FastMath.PI/2, new Vector3f(1,0,0));
eso3.setLocalRotation(q);
eso3.setModelBound(new BoundingBox());
eso3.updateModelBound();
eso3.setLocalTranslation(new Vector3f(-20,-22.5f,-70));
eso3.updateModelBound();
eso3.setRenderState(ts2);

StaticPhysicsNode staticNodeEso3 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNodeEso3);
staticNodeEso3.attachChild(eso3);
staticNodeEso3.generatePhysicsGeometry();

//eso4

Cylinder eso4= new Cylinder("eso4",20,20,2,10);
q.fromAngleAxis(FastMath.PI/2, new Vector3f(1,0,0));
eso4.setLocalRotation(q);
eso4.setModelBound(new BoundingBox());
eso4.updateModelBound();
eso4.setLocalTranslation(new Vector3f(0,-22.5f,-70));
eso4.updateModelBound();
eso4.setRenderState(ts2);

StaticPhysicsNode staticNodeEso4 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNodeEso4);
staticNodeEso4.attachChild(eso4);
staticNodeEso4.generatePhysicsGeometry();

//eso5

Cylinder eso5= new Cylinder("eso4",20,20,2,10);
q.fromAngleAxis(FastMath.PI/2, new Vector3f(1,0,0));
eso5.setLocalRotation(q);
eso5.setModelBound(new BoundingBox());
eso5.updateModelBound();
eso5.setLocalTranslation(new Vector3f(18,-22.5f,-64));
eso5.updateModelBound();
eso5.setRenderState(ts2);
```

```
StaticPhysicsNode staticNodeEso5 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNodeEso5);
staticNodeEso5.attachChild(eso5);
staticNodeEso5.generatePhysicsGeometry();

//eso6

Cylinder eso6= new Cylinder("eso4",20,20,2,10);
q.fromAngleAxis(FastMath.PI/2, new Vector3f(1,0,0));
eso6.setLocalRotation(q);
eso6.setModelBound(new BoundingBox());
eso6.updateModelBound();
eso6.setLocalTranslation(new Vector3f(35,-22.5f,-50));
eso6.updateModelBound();
eso6.setRenderState(ts2);

StaticPhysicsNode staticNodeEso6 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNodeEso6);
staticNodeEso6.attachChild(eso6);
staticNodeEso6.generatePhysicsGeometry();


//eso7

Cylinder eso7= new Cylinder("eso4",20,20,2,10);
q.fromAngleAxis(FastMath.PI/2, new Vector3f(1,0,0));
eso7.setLocalRotation(q);
eso7.setModelBound(new BoundingBox());
eso7.updateModelBound();
eso7.setLocalTranslation(new Vector3f(47,-22.5f,-25));
eso7.updateModelBound();
eso7.setRenderState(ts2);


StaticPhysicsNode staticNodeEso7 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNodeEso7);
staticNodeEso7.attachChild(eso7);
staticNodeEso7.generatePhysicsGeometry();

//eso8

Cylinder eso8= new Cylinder("eso4",20,20,2,10);
q.fromAngleAxis(FastMath.PI/2, new Vector3f(1,0,0));
eso8.setLocalRotation(q);
eso8.setModelBound(new BoundingBox());
eso8.updateModelBound();
eso8.setLocalTranslation(new Vector3f(46,-22.5f,0));
eso8.updateModelBound();
eso8.setRenderState(ts2);
```

```
StaticPhysicsNode staticNodeEso8 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNodeEso8);
staticNodeEso8.attachChild(eso8);
staticNodeEso8.generatePhysicsGeometry();


//eso9 start point

Cylinder eso9= new Cylinder("eso4",20,20,2,30);
float[] angles =  {0, 1.57f, 0};
eso9.getLocalRotation().fromAngles(angles);
eso9.setModelBound(new BoundingBox());
eso9.updateModelBound();
eso9.setLocalTranslation(new Vector3f(-10,-25.5f,45));
eso9.updateModelBound();
eso9.setRenderState(ts2);

StaticPhysicsNode staticNodeEso9 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNodeEso9);
staticNodeEso9.attachChild(eso9);

//cull state eso9
//in this object we don't want any side to be culled
CullState cseso9 = display.getRenderer().createCullState();
cseso9.setCullMode(CullState.CS_NONE);
eso9.setRenderState(cseso9);
staticNodeEso9.generatePhysicsGeometry();

//eso10 start point

Cylinder eso10= new Cylinder("eso4",20,20,2,30);
eso10.getLocalRotation().fromAngles(angles);
eso10.setModelBound(new BoundingBox());
eso10.updateModelBound();
eso10.setLocalTranslation(new Vector3f(-10,-21.5f,45));
eso10.updateModelBound();
eso10.setRenderState(ts2);

StaticPhysicsNode staticNodeEso10 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNodeEso10);
staticNodeEso10.attachChild(eso10);

//cull state eso10
// we don't any side of the object to be culled
CullState cseso10 = display.getRenderer().createCullState();
cseso10.setCullMode(CullState.CS_NONE);
eso10.setRenderState(cseso10);
staticNodeEso10.generatePhysicsGeometry();
```

```
//eso11 start point

Cylinder eso11= new Cylinder("eso4",20,20,2,10);
q.fromAngleAxis(FastMath.PI/2, new Vector3f(1,0,0));
eso11.setLocalRotation(q);
eso11.setModelBound(new BoundingBox());
eso11.updateModelBound();
eso11.setLocalTranslation(new Vector3f(34,-22.5f,25));
eso11.updateModelBound();
eso11.setRenderState(ts2);

StaticPhysicsNode staticNodeEso11 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNodeEso11);
staticNodeEso11.attachChild(eso11);
staticNodeEso11.generatePhysicsGeometry();

//eso12

Cylinder eso12= new Cylinder("eso4",20,20,2,10);
q.fromAngleAxis(FastMath.PI/2, new Vector3f(1,0,0));
eso12.setLocalRotation(q);
eso12.setModelBound(new BoundingBox());
eso12.updateModelBound();
eso12.setLocalTranslation(new Vector3f(-59,-22.5f,15));
eso12.updateModelBound();
eso12.setRenderState(ts2);

StaticPhysicsNode staticNodeEso12 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNodeEso12);
staticNodeEso12.attachChild(eso12);
staticNodeEso12.generatePhysicsGeometry();


//eso13

//create a cylinder
Cylinder eso13= new Cylinder("eso4",20,20,2,10);
q.fromAngleAxis(FastMath.PI/2, new Vector3f(1,0,0));
eso13.setLocalRotation(q);
eso13.setModelBound(new BoundingBox());
eso13.updateModelBound();
eso13.setLocalTranslation(new Vector3f(-67,-22.5f,-8));
eso13.updateModelBound();
eso13.setRenderState(ts2);

StaticPhysicsNode staticNodeEso13 = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNodeEso13);
staticNodeEso13.attachChild(eso13);
staticNodeEso13.generatePhysicsGeometry();
```

```
    //eso14

    Cylinder eso14= new Cylinder("eso4",20,20,2,10);
    q.fromAngleAxis(FastMath.PI/2, new Vector3f(1,0,0));
    eso14.setLocalRotation(q);
    eso14.setModelBound(new BoundingBox());
    eso14.updateModelBound();
    eso14.setLocalTranslation(new Vector3f(-64,-22.5f,-30));
    eso14.updateModelBound();
    eso14.setRenderState(ts2);

    StaticPhysicsNode staticNodeEso14 = getPhysicsSpace().createStaticNode();
    rootNode.attachChild(staticNodeEso14);
    staticNodeEso14.attachChild(eso14);
    staticNodeEso14.generatePhysicsGeometry();

    //eso15

    Cylinder eso15= new Cylinder("eso4",20,20,2,10);
    q.fromAngleAxis(FastMath.PI/2, new Vector3f(1,0,0));
    eso15.setLocalRotation(q);
    eso15.setModelBound(new BoundingBox());
    eso15.updateModelBound();
    eso15.setLocalTranslation(new Vector3f(-45,-22.5f,-58));
    eso15.updateModelBound();
    eso15.setRenderState(ts2);

    StaticPhysicsNode staticNodeEso15 = getPhysicsSpace().createStaticNode();
    rootNode.attachChild(staticNodeEso15);
    staticNodeEso15.attachChild(eso15);
    staticNodeEso15.generatePhysicsGeometry();

}



private void setupMoto(){

    // "car"

    // car texture
    URL orofit;
    orofit=SimpleSimple.class.
          getClassLoader().getResource(
          "orofi.jpg");

    // Get a TextureState
    TextureState orof=display.getRenderer().createTextureState();
    // Use the TextureManager to load a texture
```

```
Texture or1 = TextureManager.loadTexture(orofit,
        Texture.MM_LINEAR,
        Texture.FM_LINEAR);
// Assign the texture to the TextureState
orof.setTexture(or1);

URL par;
par=SimpleSimple.class.
        getClassLoader().getResource(
        "parath.png");
// Get a TextureState
TextureState tplc=display.getRenderer().createTextureState();
// Use the TextureManager to load a texture
Texture tp1 = TextureManager.loadTexture(par,
        Texture.MM_LINEAR,
        Texture.FM_LINEAR);
// Assign the texture to the TextureState
tplc.setTexture(tp1);

// bottom part of the car
h = new Box("mymovingbox", new Vector3f(0, 2, 0), 4f, 2f, 2f);
h.setModelBound(new BoundingBox());
h.updateModelBound();
h.setRenderState(orof);

// top part of the car
topBox = new Box("mymovingbox", new Vector3f(0, 5, 0), 2.5f, 1.5f, 1.5f);
topBox.setModelBound(new BoundingBox());
topBox.updateModelBound();
topBox.setRenderState(tplc);

// Make the car node and give it children
car = new Node("car");
car.attachChild(h);
car.attachChild(topBox);

// as the "car" should move we create a dynamic_physics node
dynamicNode = getPhysicsSpace().createDynamicNode();
//attach the node to the root Node
rootNode.attachChild(dynamicNode);
// car.setLocalTranslation(new Vector3f (-15,0,50));

// create a Node to place the camera on
cam_char = new Node("char node");

//attach cam_char to the dynamic node
dynamicNode.attachChild(cam_char);

//attach car to the chase camera(cam_char)
cam_char.attachChild(car);
```

```
    dynamicNode.setLocalTranslation(new Vector3f(-25,-27.5f,60));
    dynamicNode.generatePhysicsGeometry();
    showPhysics=false;
}



private void setupSkybox() {

    // create skybox
    skybox = new Skybox("skybox", 135, 30, 135);

    //assign texture for the north part of the skybox
    Texture north = TextureManager.loadTexture(
            SimpleSimple.class.getClassLoader().getResource(
            "CAAJRYPD.jpg"),
            Texture.MM_LINEAR,
            Texture.FM_LINEAR);

    //assign texture for the south part of the skybox
    Texture south = TextureManager.loadTexture(
            SimpleSimple.class.getClassLoader().getResource(
            "CAAJRYPD.jpg"),
            Texture.MM_LINEAR,
            Texture.FM_LINEAR);
    //assign texture for the east part of the skybox
    Texture east = TextureManager.loadTexture(
            SimpleSimple.class.getClassLoader().getResource(
            "CAAJRYPD.jpg"),
            Texture.MM_LINEAR,
            Texture.FM_LINEAR);
    //assign texture for the west part of the skybox
    Texture west = TextureManager.loadTexture(
            SimpleSimple.class.getClassLoader().getResource(
            "CAAJRYPD_128.jpg"),
            Texture.MM_LINEAR,
            Texture.FM_LINEAR);
    //assign texture for the top part of the skybox
    Texture up = TextureManager.loadTexture(
            SimpleSimple.class.getClassLoader().getResource(
            "clouds.png"),
            Texture.MM_LINEAR,
            Texture.FM_LINEAR);
    //assign texture for the bottom part of the skybox
    Texture down = TextureManager.loadTexture(
            SimpleSimple.class.getClassLoader().getResource(
            "marble128.png"),
            Texture.MM_LINEAR,
```

```
        Texture.FM_LINEAR);

skybox.setTexture(Skybox.NORTH, north);
skybox.setTexture(Skybox.WEST, west);
skybox.setTexture(Skybox.SOUTH, south);
skybox.setTexture(Skybox.EAST, east);
skybox.setTexture(Skybox.UP, up);
skybox.setTexture(Skybox.DOWN, down);
skybox.preloadTextures();

rootNode.attachChild(skybox);

//create invisible wall around the skybox so our car want fall through
// the stage

//pleura apenanti apo to start point

StaticPhysicsNode staticNodeSky = getPhysicsSpace().createStaticNode();
rootNode.attachChild( staticNodeSky );
PhysicsBox skywest = staticNodeSky.createBox("skywest");
skywest.getLocalScale().set(300f,0.5f,300f);
//apply a rotation
Quaternion qsky = new Quaternion();
qsky.fromAngleAxis(FastMath.PI/2, new Vector3f(1,0,0));
skywest.setLocalRotation(qsky);
skywest.setLocalTranslation(new Vector3f(0,-2,-135));

//pleura start point

StaticPhysicsNode staticNodeSkyE = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNodeSkyE);
PhysicsBox skyeast = staticNodeSkyE.createBox("sky east");
skyeast.getLocalScale().set(300,0.5f,300);
skyeast.setLocalRotation(qsky);
skyeast.setLocalTranslation(new Vector3f(0,-2,135));

//piso apo to start point

StaticPhysicsNode staticNodeSkyN = getPhysicsSpace().createStaticNode();
rootNode.attachChild(staticNodeSkyN);
PhysicsBox skynorth = staticNodeSkyN.createBox("sky east");
skynorth.getLocalScale().set(300,0.5f,570);
Quaternion qsky2 = new Quaternion();
qsky2.fromAngleAxis(FastMath.PI/2, new Vector3f(1,0,0));
skynorth.setLocalRotation(qsky2);
float [] rotsky = { 0,0,FastMath.PI/2};
skynorth.setLocalRotation(new Quaternion(rotsky));
skynorth.setLocalTranslation(new Vector3f(-135,-2,135));

//apenanti apo to start point
```

```
    StaticPhysicsNode staticNodeSkyS = getPhysicsSpace().createStaticNode();
    rootNode.attachChild(staticNodeSkyS);
    PhysicsBox skysouth = staticNodeSkyS.createBox("sky east");
    skysouth.getLocalScale().set(300,0.5f,570);
    //Quaternion qsky2 = new Quaternion();
    qsky2.fromAngleAxis(FastMath.PI/2, new Vector3f(1,0,0));
    skysouth.setLocalRotation(qsky2);
    //float [] rotsky = { 0,0,FastMath.PI/2};
    skysouth.setLocalRotation(new Quaternion(rotsky));
    skysouth.setLocalTranslation(new Vector3f(135,-2,135));

    // we don't need to render the back face of triangles so we cull them
    //that save us time and poweer
    CullState cs = display.getRenderer().createCullState();
    cs.setCullMode(CullState.CS_BACK);
    rootNode.setRenderState(cs);

}

private void setupChaseCam(){

    //set the chase camera
    chaser = new ChaseCamera(cam, cam_char);

}


private void setupInput(){


    //set up handles properties
    HashMap<String, Object> handlerProps = new HashMap<String, Object>();

    handlerProps.put(ThirdPersonHandler.PROP_DOGRADUAL, "true");
    //when true the controlled target will do turns by moving forward
        and turning at the same time.
    //when false a turn will cause immediate rotation to the given angle

    handlerProps.put(ThirdPersonHandler.PROP_TURNSPEED, ""+(2f *
   FastMath.PI));
    //turn speed

    handlerProps.put(ThirdPersonHandler.PROP_LOCKBACKWARDS, "false");
    //if true backwards movements will not cause the target to rotate around to
        point backward

    handlerProps.put(ThirdPersonHandler.PROP_CAMERAALIGNEDMOVE,
   "false");
    //if true movements of the character are in relation to the current camera view
```

```
    chaser.setEnableSpring(true);
    //chaser.setEnableSpring(false);

    chaser.setTarget(dynamicNode);

    //set the camera to be stable behind an object
    chaser.setMaxDistance(70F);
    chaser.setMinDistance(70F);
    chaser.setStayBehindTarget(true);

    //create our input handler
    input = new ThirdPersonHandler(dynamicNode, cam, handlerProps);
    //moving speed
    input.setActionSpeed(50f);
}



private void setupEmpodia() {

    //create bonus spheres

    // bonus 1

    DynamicBonus1 = getPhysicsSpace().createDynamicNode();
    DynamicBonus1.setLocalTranslation(new Vector3f(50,-21.5f,20));
    rootNode.attachChild(DynamicBonus1);
    Bonus1 = new Sphere("empodio1", 20,20,2f );
    Bonus1.setModelBound(new BoundingSphere());
    Bonus1.updateModelBound();

    //set texture
    URL Empodiotexture;
    Empodiotexture=SimpleSimple.class.
          getClassLoader().getResource(
          "rainbow.jpg");
    // Get a TextureState
    TextureState sp1=display.getRenderer().createTextureState();
    // Use the TextureManager to load a texture
    Texture s = TextureManager.loadTexture(Empodiotexture,
          Texture.MM_LINEAR,
          Texture.FM_LINEAR);  //MM_LINEAR and FM_LINEAR let us know hot
   to filter the texture

    // Assign the texture to the TextureState
    sp1.setTexture(s);
    Bonus1.setRenderState(sp1);
    DynamicBonus1.attachChild(Bonus1);
    DynamicBonus1.generatePhysicsGeometry();
```

```
// bonus2

DynamicBonus2 = getPhysicsSpace().createDynamicNode();
DynamicBonus2.setLocalTranslation(new Vector3f(10,-23.5f,-90));
rootNode.attachChild(DynamicBonus2);
Bonus2 = new Sphere("Bonus2",20,20,2f);
Bonus2.setModelBound(new BoundingSphere());
Bonus2.updateModelBound();
Bonus2.setRenderState(sp1);
DynamicBonus2.attachChild(Bonus2);
DynamicBonus2.generatePhysicsGeometry();

//bonus3

DynamicBonus3 = getPhysicsSpace().createDynamicNode();
DynamicBonus3.setLocalTranslation(new Vector3f(-50,-23.5f,-90));
rootNode.attachChild(DynamicBonus3);
Bonus3 = new Sphere("Bonus3",20,20,2f);
Bonus3.setModelBound(new BoundingSphere());
Bonus3.updateModelBound();
Bonus3.setRenderState(sp1);
DynamicBonus3.attachChild(Bonus3);
DynamicBonus3.generatePhysicsGeometry();


//bonus4

DynamicBonus4 = getPhysicsSpace().createDynamicNode();
DynamicBonus4.setLocalTranslation(new Vector3f(-35,-23.5f,60));
rootNode.attachChild(DynamicBonus4);
Bonus4 = new Sphere("Bonus4",20,20,2f);
Bonus4.setModelBound(new BoundingSphere());
Bonus4.updateModelBound();
Bonus4.setRenderState(sp1);
DynamicBonus4.attachChild(Bonus4);
DynamicBonus4.generatePhysicsGeometry();

// create bombes spheres
//create bomba1

DynamicBomba1 = getPhysicsSpace().createDynamicNode();
DynamicBomba1.setLocalTranslation(new Vector3f(20,-23.5f,60));
rootNode.attachChild(DynamicBomba1);
Sphere Bomba1 = new Sphere("Bomba1", 20,20,2f );
Bomba1.setModelBound(new BoundingSphere());
Bomba1.updateModelBound();
URL Bobmatexture;
Bobmatexture=SimpleSimple.class.
      getClassLoader().getResource(
```

```
        "borderkitrino.png");
// Get a TextureState
TextureState bom1=display.getRenderer().createTextureState();
// Use the TextureManager to load a texture
Texture b1 = TextureManager.loadTexture(Bobmatexture,
        Texture.MM_LINEAR,
        Texture.FM_LINEAR);  //MM_LINEAR and FM_LINEAR let us know hot
to filter the texture
// Assign the texture to the TextureState
bom1.setTexture(b1);
Bomba1.setRenderState(bom1);
DynamicBomba1.attachChild(Bomba1);
DynamicBomba1.generatePhysicsGeometry();


//create bomba2

DynamicBomba2 = getPhysicsSpace().createDynamicNode();
DynamicBomba2.setLocalTranslation(new Vector3f(-60,-23.5f,-60));
rootNode.attachChild(DynamicBomba2);
Sphere Bomba2 = new Sphere("Bomba2", 20,20,2f );
Bomba2.setModelBound(new BoundingSphere());
Bomba2.updateModelBound();
Bomba2.setRenderState(bom1);
DynamicBomba2.attachChild(Bomba2);
DynamicBomba2.generatePhysicsGeometry();

//create bomba3

DynamicBomba3 = getPhysicsSpace().createDynamicNode();
DynamicBomba3.setLocalTranslation(new Vector3f(55,-23.5f,-50));
rootNode.attachChild(DynamicBomba3);
Sphere Bomba3 = new Sphere("Bomba3", 20,20,2f );
Bomba3.setModelBound(new BoundingSphere());
Bomba3.updateModelBound();
Bomba3.setRenderState(bom1);
DynamicBomba3.attachChild(Bomba3);
DynamicBomba3.generatePhysicsGeometry();


//create bomba4

DynamicBomba4 = getPhysicsSpace().createDynamicNode();
DynamicBomba4.setLocalTranslation(new Vector3f(-90,-23.5f,-20));
rootNode.attachChild(DynamicBomba4);
Sphere Bomba4 = new Sphere("Bomba4", 20,20,2f );
Bomba4.setModelBound(new BoundingSphere());
Bomba4.updateModelBound();
Bomba4.setRenderState(bom1);
DynamicBomba4.attachChild(Bomba4);
```

```
    DynamicBomba4.generatePhysicsGeometry();

    //create bomba5

    DynamicBomba5 = getPhysicsSpace().createDynamicNode();
    DynamicBomba5.setLocalTranslation(new Vector3f(-55,-23.5f,35));
    rootNode.attachChild(DynamicBomba5);
    Sphere Bomba5 = new Sphere("Bomba5", 20,20,2f );
    Bomba5.setModelBound(new BoundingSphere());
    Bomba5.updateModelBound();
    Bomba5.setRenderState(bom1);
    DynamicBomba5.attachChild(Bomba5);
    DynamicBomba5.generatePhysicsGeometry();

     // create box for the finish line
    staticNodeFinish = getPhysicsSpace().createStaticNode();
    // create texture
    URL finitoLoc;
    finitoLoc=SimpleSimple.class
          .getClassLoader().getResource(
          "BlackAndWhite.jpg");
    // Get a TextureState
    TextureState fbox=display.getRenderer().createTextureState();
    // Use the TextureManager to load a texture
    Texture fi = TextureManager.loadTexture(finitoLoc,
           Texture.MM_LINEAR,
           Texture.FM_LINEAR);  //MM_LINEAR and FM_LINEAR let us know hot
   to filter the texture

    // Assign the texture to the TextureState
    fbox.setTexture(fi);

    Box FinishBox = new Box("finitoB", new Vector3f(0,-17,0),1,1,1);
    FinishBox.getLocalScale().set(0.3f,1f,17);
    FinishBox.setModelBound(new BoundingBox());
    FinishBox.updateModelBound();

    // render state for finish Box
      FinishBox.setRenderState(fbox);
    // showPhysics = true;

     staticNodeFinish.attachChild(FinishBox);

}

private void BoxTermatismou(){

    // display the finishing line box

     staticNodeFinish.setLocalTranslation(new Vector3f(-13,-7.5f,60));
```

```
    rootNode.attachChild(staticNodeFinish);
    staticNodeFinish.generatePhysicsGeometry();
}



protected void Countdown(){

   //set up timer

   //type format of timer
   event = Calendar.getInstance();  //returns a calendar object whose time fiels
                           //have been initialized with the current date and time

   nowDate = event.getTime();
   long now = nowDate.getTime();
   end = now + 60000;
   System.out.println(end);

   //text to be dispalyed and it's location
   completionText = new Text("textNode", "MISSION COMPLETE");
   completionText.setLocalTranslation(new Vector3f(300,300,0));

   //build the timer
   drawTimer();


}


protected void LivesAndBonus(){

   //set initial number of players lives
   lives =3;
   //display lives on sceen
   Text LivesText = new Text("textNode2", "LIVES: " +lives);
   LivesText.setLocalTranslation(new Vector3f(100,100,1));
   //attach LivesText to fpsNode
   //fpsNode is always on top of every other object in the sceen
   fpsNode.attachChild(LivesText);

   //Bonus

   bonus =0;
   BonusText = new Text("Btxt","BONUS: "+ bonus);
   BonusText.setLocalTranslation(new Vector3f(480,100,1));
   fpsNode.attachChild(BonusText);

}
```

```
// @Override


    protected void simpleUpdate() {
        int tt = 0;
        int i;
        boolean f= false;
        boolean t= true;
        reDrawTimer = true;

        //update every frame
        chaser.update(tpf);

        //check if "car" falls off the terrain
        if (dynamicNode.getWorldTranslation().y < -100){
            dynamicNode.clearDynamics();
            //place car back on top of the terrain
            dynamicNode.getLocalTranslation().set(0,-5,0);

            // substrac-reduce life
            lives=lives-1;
            //update number of lifes on sceen
            fpsNode.detachChild(LivesText);
            fpsNode.detachChildNamed("textNode2");
            LivesText = new Text("textNode2", "LIVES: " +lives);
            LivesText.setLocalTranslation(new Vector3f(100,100,1));
            fpsNode.attachChild(LivesText);

        }

        if (lives<0) {
            //to diaplay lives 0 and not a negative number
            fpsNode.detachChild(LivesText);
            fpsNode.detachChildNamed("textNode");
            LivesText = new Text("textNode2", "LIVES:  0" );
            LivesText.setLocalTranslation(new Vector3f(100,100,1));
            fpsNode.attachChild(LivesText);
            fpsNode.detachChild(completionText);
            fpsNode.detachChild(EndLivesText);

            // diplay condision messages
            completionText = new Text("textNode", "MISSION COMPLETE");
            completionText.setLocalTranslation(new Vector3f(300,300,0));
            fpsNode.attachChild(completionText);

            EndLivesText = new Text("textNode", "YOU ARE DEAD !!!");
            EndLivesText.setLocalTranslation(new Vector3f(500,300,0));
            fpsNode.attachChild(EndLivesText);

            // play again. Give the oportunity to contuniou the game
```

```
            fpsNode.detachChildNamed("retxt");
            Text restartTxt = new Text("retxt", "PLAY AGAIN (Y/N)?" );
            restartTxt.setLocalTranslation(new Vector3f(400,500,0));
            fpsNode.attachChild(restartTxt);
            //disable the input hander
            input.setEnabled(f);

            //check if y is pressed

KeyBindingManager.getKeyBindingManager().set("check",KeyInput.KEY_Y);
            if
(KeyBindingManager.getKeyBindingManager().isValidCommand("check",true)){
                lives = 3;
                //refresh player's statistics
                fpsNode.detachChild(EndLivesText);
                fpsNode.detachChild(completionText);
                fpsNode.detachChild(restartTxt);
                fpsNode.detachChildNamed("textNode");
                //enable the input handler
                input.setEnabled(t);

                fpsNode.detachChildNamed("textNode2");
                LivesText = new Text("textNode2", "LIVES: " +lives);
                LivesText.setLocalTranslation(new Vector3f(100,100,1));
                fpsNode.attachChild(LivesText);

            }


        }

        //check if I run out of time

        event = Calendar.getInstance();
        nowDate = event.getTime();
        long now = nowDate.getTime();
        //    if ((now >=end) && (finished= false)){
        if (now >=end){
            reDrawTimer = false;
            fpsNode.detachChildNamed("textNode");
            fpsNode.detachChild(EndLivesText);
            fpsNode.detachChild(EndTimeText);

            // check if all bonus are collected
            if (finished == false){
                fpsNode.attachChild(completionText);
                EndTimeText = new Text("textTime", "RUN OUT OF TIME   !!");
                EndTimeText.setLocalTranslation(new Vector3f(500,300,0));
                fpsNode.attachChild(EndTimeText);
                fpsNode.detachChild(restartTxt);
```

```
            restartTxt = new Text("retxt", "PLAY AGAIN (Y/N)?" );
            restartTxt.setLocalTranslation(new Vector3f(400,500,0));
            fpsNode.attachChild(restartTxt);
            input.setEnabled(f);



KeyBindingManager.getKeyBindingManager().set("check2",KeyInput.KEY_Y);
            if
(KeyBindingManager.getKeyBindingManager().isValidCommand("check2",true)){
                input.setEnabled(t);
                System.out.println("patithike");
                fpsNode.detachChild(EndLivesText);
                fpsNode.detachChild(completionText);
                fpsNode.detachChild(restartTxt);
                // fpsNode.detachChild(restartTxt);
                fpsNode.detachChild(EndTimeText);
                fpsNode.detachChildNamed("retxt");
                fpsNode.detachChildNamed("textNode2");
                LivesText= new Text("textNode2", "LIVES: " +lives);
                LivesText.setLocalTranslation(new Vector3f(100,100,1));
                fpsNode.attachChild(LivesText);

                 lives= 3;
                 drawTimer();
                Countdown();
              }
          }

       }

      // if bonus are not all collected continiou updating the timer
     if (finished == false){
         if (reDrawTimer) {
             drawTimer();
          }
     }


       // check for intersection
       //update all Bounds of the scene
       rootNode.updateWorldBound();

       // see if my car intersects with any Bonus sphere
       if (DynamicBonus1.getWorldBound().intersects(car.getWorldBound())){
          System.out.println("OUCH");
          // make bonus invisible
          DynamicBonus1.setLocalTranslation(new Vector3f(0,-27.5f,0));
          DynamicBonus1.generatePhysicsGeometry();
          //detach bonus from the scene
```

```
    DynamicBonus1.detachAllChildren();
    rootNode.detachChild(DynamicBonus1);
    fpsNode.detachChild(BonusText);
    dynamicNode.generatePhysicsGeometry();
    // increse number of bonus collected
    bonus++;
    // update bonus text on screen
    BonusText = new Text("Btxt","BONUS: "+ bonus);
    BonusText.setLocalTranslation(new Vector3f(480,100,1));
    fpsNode.attachChild(BonusText);

    if (bonus==4){
       BoxTermatismou();
    }


}

// intersection Bonus2

rootNode.updateWorldBound();
if (DynamicBonus2.getWorldBound().intersects(car.getWorldBound())){
    System.out.println("Bonus2");
    DynamicBonus2.setLocalTranslation(new Vector3f(0,-27.5f,0));
    DynamicBonus2.generatePhysicsGeometry();
    DynamicBonus2.detachAllChildren();
    rootNode.detachChild(DynamicBonus2);
    fpsNode.detachChild(BonusText);
    //  check[1]=true;
    bonus++;
    BonusText = new Text("Btxt","BONUS: "+ bonus);
    BonusText.setLocalTranslation(new Vector3f(480,100,1));
    fpsNode.attachChild(BonusText);
    if (bonus==4){
       BoxTermatismou();
    }

}

//intersecton Bonus3
rootNode.updateWorldBound();
if (DynamicBonus3.getWorldBound().intersects(car.getWorldBound())){
    System.out.println("Bonus3");
    DynamicBonus3.setLocalTranslation(new Vector3f(0,-27.5f,0));
    DynamicBonus3.generatePhysicsGeometry();
    DynamicBonus3.detachAllChildren();
    rootNode.detachChild(DynamicBonus3);
    fpsNode.detachChild(BonusText);
    // check[2] = true;
    bonus++;
```

```
    BonusText = new Text("Btxt","BONUS: "+ bonus);
    BonusText.setLocalTranslation(new Vector3f(480,100,1));
    fpsNode.attachChild(BonusText);
    if (bonus==4){
       BoxTermatismou();
    }

}


// intersection bonus4
rootNode.updateWorldBound();
if (DynamicBonus4.getWorldBound().intersects(car.getWorldBound())){
    System.out.println("Bonus4");
    DynamicBonus4.setLocalTranslation(new Vector3f(0,-27.5f,0));
    DynamicBonus4.generatePhysicsGeometry();
    DynamicBonus4.detachAllChildren();
    rootNode.detachChild(DynamicBonus4);
    fpsNode.detachChild(BonusText);
    bonus++;
    // check[3]=true;
    BonusText = new Text("Btxt","BONUS: "+ bonus);
    BonusText.setLocalTranslation(new Vector3f(480,100,1));
    fpsNode.attachChild(BonusText);
    System.out.println("to bonous exei timi "+bonus);

    if (bonus==4){
       BoxTermatismou();
    }

}


// loose live when hittign bomba 1

if (DynamicBomba1.getWorldBound().intersects(car.getWorldBound())){
    System.out.println("bomba1");
    // change car location
    dynamicNode.setLocalTranslation(new Vector3f(25,-21.5f,60));
    dynamicNode.generatePhysicsGeometry();
    // substrac-reduce life
    lives=lives-1;
    //update number of lifes on sceen
    fpsNode.detachChild(LivesText);
    fpsNode.detachChildNamed("textNode2");
    LivesText = new Text("textNode2", "LIVES: " +lives);
    LivesText.setLocalTranslation(new Vector3f(100,100,1));
    fpsNode.attachChild(LivesText);

}
```

```
// loose live when hittign bomba 2

if (DynamicBomba2.getWorldBound().intersects(car.getWorldBound())){
   System.out.println("bomba2");
   dynamicNode.setLocalTranslation(new Vector3f(-55,-21.5f,-60));
   dynamicNode.generatePhysicsGeometry();
   // substrac-reduce life
   lives=lives-1;

   //update number of lifes on sceen
   fpsNode.detachChild(LivesText);
   fpsNode.detachChildNamed("textNode2");
   LivesText = new Text("textNode2", "LIVES: " +lives);
   LivesText.setLocalTranslation(new Vector3f(100,100,1));
   fpsNode.attachChild(LivesText);

}

// loose live when hittign bomba 3

if (DynamicBomba3.getWorldBound().intersects(car.getWorldBound())){
   System.out.println("bomba3");
   dynamicNode.setLocalTranslation(new Vector3f(65,-21.5f,-45));
   dynamicNode.generatePhysicsGeometry();
   // substrac-reduce life
   lives=lives-1;
   //update number of lifes on sceen
   fpsNode.detachChild(LivesText);
   fpsNode.detachChildNamed("textNode2");
   LivesText = new Text("textNode2", "LIVES: " +lives);
   LivesText.setLocalTranslation(new Vector3f(100,100,1));
   fpsNode.attachChild(LivesText);

}

// loose live when hittign bomba 4

if (DynamicBomba4.getWorldBound().intersects(car.getWorldBound())){
   System.out.println("bomba4");
   dynamicNode.setLocalTranslation(new Vector3f(-80,-21.5f,-15));
   dynamicNode.generatePhysicsGeometry();
   // substrac-reduce life
   lives=lives-1;
   //update number of lifes on sceen
   fpsNode.detachChild(LivesText);
   fpsNode.detachChildNamed("textNode2");
   LivesText = new Text("textNode2", "LIVES: " +lives);
   LivesText.setLocalTranslation(new Vector3f(100,100,1));
```

```
            fpsNode.attachChild(LivesText);

        }

        // loose live when hittign bomba 5

        if (DynamicBomba5.getWorldBound().intersects(car.getWorldBound())){
            System.out.println("bomba5");
            dynamicNode.setLocalTranslation(new Vector3f(-45,-21.5f,40));
            dynamicNode.generatePhysicsGeometry();
            // substrac-reduce life
            lives=lives-1;
            //update number of lifes on sceen
            fpsNode.detachChild(LivesText);
            fpsNode.detachChildNamed("textNode2");
            LivesText = new Text("textNode2", "LIVES: " +lives);
            LivesText.setLocalTranslation(new Vector3f(100,100,1));
            fpsNode.attachChild(LivesText);

        }

        // check for winning

        if (staticNodeFinish.getWorldBound().intersects(car.getWorldBound())){
            System.out.println("END");
            fpsNode.detachChild(winText);
            winText = new Text("winTxt","CONGRATULATIONS...YOU WON!!");
            winText.setLocalTranslation(new Vector3f(400,600,0));
            fpsNode.attachChild(winText);

            input.setEnabled(f);
            finished = true;
            fpsNode.detachChild(EndTimeText);
            fpsNode.detachChild(completionText);

             /*
             // play again
            fpsNode.detachChildNamed("retxt");
            Text restartTxt = new Text("retxt", "PLAY AGAIN (Y/N)?" );
            restartTxt.setLocalTranslation(new Vector3f(400,500,0));
            fpsNode.attachChild(restartTxt);
          //  input.setEnabled(f);


                  //check if y is pressed
            KeyBindingManager.getKeyBindingManager().set("che3",KeyInput.KEY_Y);
            if
(KeyBindingManager.getKeyBindingManager().isValidCommand("che3",true)){
                    input.setEnabled(t);
                       Countdown();
```

```
            bonus=0;
          System.out.println("elenxos gia to y");

           lives = 3;
           fpsNode.detachChild(EndLivesText);
           fpsNode.detachChild(completionText);
           fpsNode.detachChild(restartTxt);
           fpsNode.detachChildNamed("textNode");


           fpsNode.detachChildNamed("textNode2");
           LivesText = new Text("textNode2", "LIVES: " +lives);
           LivesText.setLocalTranslation(new Vector3f(100,100,1));
           fpsNode.attachChild(LivesText);
           fpsNode.detachChild(winText);
           bonus=0;
           fpsNode.detachChild(BonusText);
            BonusText = new Text("Btxt","BONUS: "+ bonus);
            fpsNode.attachChild(BonusText);
            rootNode.detachChild(Bonus1);

        }
     */
    }

    // set the camera not to view underneath the ground

    Vector3f camloc = chaser.getCamera().getLocation();
    camloc.y= -9f;
    chaser.getCamera().setLocation(camloc);



}

private void drawTimer() {

    StringBuilder sb = new StringBuilder();
    nowDate = event.getTime();
    long now = nowDate.getTime();
    //set the way-format that timer is going to be displayed
    sb.append(":");
    sb.append(twoDecimals.format((end - now)/1000));
    final Text timerText = new Text("textNode", sb.toString());
    timerText.setLocalTranslation(new Vector3f(850,100,1));

    //update timer on the sceen
    fpsNode.detachChildNamed("textNode");
    fpsNode.attachChild(timerText);
```

```
    }
    protected void reinit() {
    }
    protected void cleanup() {
    }
}
```

Οδηγίες Παιχνιδιού – Σκοπός.

Πάνω στην πίστα μας υπάρχουν 5 βόμβες και 4 μπόνους. Σκοπός του παιχνιδιού είναι να κάνουμε τον γύρο της πίστας μέσα σε 60 δευτερόλεπτα μαζεύοντας όλα τα μπόνους . Στην διάθεση μας έχουμε τρεις ζωές. Κάθε φορά που το αυτοκινητάκι χτυπάει πάνω σε μια βόμβα χάνουμε αυτόματα και μια ζωή. Οι βόμβες συμβολίζονται με τις κιτρινόμαυρες σφαίρες ενώ τα μπόνους με τις χρωματιστές σφαίρες. Το τοίχος τερματισμού θα εμφανιστεί αποκλειστικά και μόνο αν έχουν μαζευτεί όλα τα μπόνους. Τα μπόνους μαζεύονται απλός περνώντας από πάνω τους. Η σειρά που θα μαζέψουμε τα μπόνους δεν έχει σημασία. Αν χάσουμε όλες τις ζωές μας τότε εμφανίζονται στην οθόνη μας τα ακόλουθα μηνύματα: “PLAY AGAIN (Y/N)?” και “ MISSION COMPLETE..YOU ARE DEAD!!” ενώ αν τελειώσει ο χρόνος μας εμφανίζονται τα μηνύματα: “PLAY AGAIN (Y/N)?” και “MISSION COMPLETE..RUN OUT OF TIME”. Και στις δύο περιπτώσεις το αυτοκίνητο ακινητοποιείται και απενεργοποιούνται όλες του οι κινήσεις. Πατώντας το πλήκτρο Y το παιχνίδι μας δίνει την δυνατότητα να συνεχίσουμε ακριβώς από το σημείο που σταματήσαμε και με τις ίδιες συνθήκες, ανανεώνοντας τον χρόνο ή της ζωές αντίστοιχα. Τις κινήσεις του αυτοκίνητου τις χειριζόμαστε με τα πλήκτρα, W για να κινηθούμε προς τα μπροστά, A για να στρίψουμε προς τα αριστερά, και D για να στρίψουμε προς τα δεξιά. Αν θέλουμε να κινηθούμε προς τα πίσω θα πρέπει να κάνουμε στροφή 180^{o} . Επίσης με τα πλήκτρα Q και E μπορούμε να κινήσουμε το έδαφος σε σχέση με το όχημα μας, αριστερά και δεξιά αντίστοιχα. Ακόμα πατώντας το F1 έχουμε την δυνατότητα να τραβήξουμε ένα screenshot.

Βιβλιογραφία

http://www.jmonkeyengine.com/ - jmonkeyengine.com Home

http://www.jmonkeyengine.com/doc/ - jME API

http://www.jmonkeyengine.com/jmeforum/ - jMonkey Engine

http://www.jmonkeyengine.com/wiki/doku.php - jME Wiki

http://www.jmonkeyengine.com/wiki/doku.php?id=starter:jme_wiki - starter jME_wiki

http://lwjgl.org/ - lwjgl.org – Home of the Lightweight Java Game Library

http://www.opengl.org/about/overview/ - OpenGL Overview

http://www.ode.org/ - Open Dynamic Engine – home

http://odejava.org/OdejavaIntro - Odejava: OdejavaIntro

http://pisa.ucsd.edu/cse125/2006/Papers/Introduction_to_3d_Game_Engines.pdf

http://www.flipcode.com/articles/article_frustumculling.shtml - article_frustumculling

http://jme-physics.sourceforge.net/ - jME Physics System

http://fortworthjug.org:8090/fwjug/resources/topics/IntroTojME.pdf

http://ode.org/ode-latest-userguide.html#sec_12_13_0 - Open Dynamics Engine

http://en.wikipedia.org/wiki/Game_engine - Game engine - Wikipedia