Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Εφαρμοσμένης Πληροφορικής & Πολυμέσων

Πτυχιακή Εργασία

Τίτλος:
Δημιουργία δισδιάστατων παιχνιδιών σε Java

Κάντι Βίκτωρας 2818

Επιβλέπων Καθηγητής : Παπαδάκης Νικόλαος
Επιτροπή Αξιολόγησης : Παπαδάκης Νικόλαος, Ρουσάκης Ιωάννης, Τσάμης Γεώργιος
Ημερομηνία Παρουσίασης : 03 – 02 – 2015

Abstract of the final year project in English

This project combines a game library with an environment which aim to simplify the development of 2D computer games using Java. It introduces a new way to describe and build games in a way simpler for the user to understand and requiring less code from his part. It provides many ready objects and algorithms frequently used in game development which are widely customizable and adjust to every type of game.

The environment consists of three editors for creating sprites, animations, particle systems, lights, bounds and combines them to synthesize scenes. All the exported files from this environment can be read from the library and loaded to game objects to create interactive scenes. The library provides all the functions necessary to handle those objects and covers a wide variety of options.

It's a useful tool for indie game developers who want to skip the implementation details needed from almost every game and have quick results. It can be used to fully automate the loading, rendering and running of game scenes along with other functionalities such as collisions, depth layering, user input, sound management, networking, neutral entity control as well as data structures for save game files, dialogs, high score boards and more. Using this tool we can minimize the time and effort programming consumes and focus on things that make a game interesting like plot, art and effects.

All the objects are described in commented text files which can be easily read and edited by the user. By giving freedom to the user to easily edit these objects and see the effects they make in the game, this project can be used as a learning tool as well.

Σύνοψη Εργασίας

Αυτή η εργασία συνδυάζει μια βιβλιοθήκη παιχνιδιών με ένα περιβάλλον που αποσκοπούν στην απλούστευση της ανάπτυξης των δις-διάστατων ηλεκτρονικών παιχνιδιών με τη χρήση της Java . Εισάγει ένα νέο τρόπο για να περιγράφει και να δομεί παιχνίδια πιο απλό για τον χρήστη να κατανοήσει που απαιτεί λιγότερο κώδικα από την πλευρά του . Παρέχει πολλά έτοιμα αντικείμενα και αλγόριθμους που χρησιμοποιούνται συχνά στην ανάπτυξη παιχνιδιών που είναι ευρέως παραμετροποιήσημα και προσαρμόζονται σε κάθε τύπο παιχνιδιού .

Το περιβάλλον αποτελείται από τρεις συντάκτες για τη δημιουργία sprite , animation, συστήματα σωματιδίων ,φωτισμούς , bounds και τα συνδυάζει για να συνθέσει σκηνές . Όλα τα αρχεία που εξάγονται από αυτό το περιβάλλον μπορούν να διαβαστούν από τη βιβλιοθήκη και φορτώνονται σε αντικείμενα του παιχνιδιού για να δημιουργήσουν διαδραστικές σκηνές . Η βιβλιοθήκη παρέχει όλες τις απαραίτητες λειτουργίες για να χειριστεί αυτά τα αντικείμενα και καλύπτει μια ευρεία ποικιλία επιλογών.

Είναι ένα χρήσιμο εργαλείο για τους indie προγραμματιστές παιχνιδιών που θέλουν να παρακάμψουν τις λεπτομέρειες υλοποίησης που απαιτούνται από σχεδόν κάθε παιχνίδι και να έχουν γρήγορα αποτελέσματα . Μπορεί να χρησιμοποιηθεί για να αυτοματοποιήσει πλήρως τη φόρτωση , το rendering και το τρέξιμο των σκηνών μαζί με άλλες λειτουργίες όπως συγκρούσεις , βάθος διαστρωμάτωσης, είσοδο χρήστη, διαχείριση ήχου, δικτύωση , έλεγχο ουδέτερων αντικειμένων, καθώς και δομές δεδομένων για την αποθήκευση προόδου , διαλόγους , πίνακες βαθμολογίας και άλλα . Χρησιμοποιώντας αυτό το εργαλείο μπορούμε να ελαχιστοποιήσουμε το χρόνο που απαιτεί ο προγραμματισμός και να επικεντρωθούμε σε πράγματα που κάνουν ένα παιχνίδι ενδιαφέρον όπως η πλοκή, η τέχνη και τα εφέ.

Όλα τα αντικείμενα περιγράφονται σε αρχεία κειμένου που μπορούν εύκολα να διαβαστούν και να συνταχθούν από το χρήστη . Δίνοντας την ελευθερία στον χρήστη να επεξεργάζεται εύκολα τα αντικείμενα αυτά και να βλέπει άμεσα τα αποτελέσματα που επιφέρουν στο παιχνίδι , αυτή η εργασία μπορεί να χρησιμοποιηθεί και ως εργαλείο εκμάθησης.

Πίνακας Περιεχομένων

Πίνακας εικόνων

1 Εισαγωγή

Η ανάπτυξη ηλεκτρονικών παιχνιδιών είναι μια διαδικασία που μπορεί να διεκπεραιωθεί απο ένα άτομο έως μια μεγάλη επιχείρηση. Στην εποχή των πρώτων παιχνιδιών το 1970, λόγω των περιορισμένων δυνατοτήτων των υπολογιστών και της χαμηλής οικονομικής απαίτησης ένας προγραμματιστής μπορούσε να αναλάβει όλη την διαδικασία υλοποίησης ενός ηλεκτρονικού παιχνιδιού. Απο τον προγραμματισμό μέχρι τα γραφικά και τον ήχο. Πλέον όμως με την ανάπτυξη της τεχνολογίας και την αύξηση απαιτήσεων των καταναλωτών είναι σχεδόν αδύνατον για ένα άτομο να ανταγωνιστεί τις μεγάλες επιχειρήσεις που αποτελούνται απο εκατοντάδες άτομα και αφιερώνουν τεράστια κεφάλαια και πολύ χρόνο στην δημιουργία παιχνιδιών. Όμως αντίθετα με ότι θα περίμενε κανείς τέτοιοι indie δημιουργοί είδαν μεγάλη ανάπτυξη τα τελευταία χρόνια καθώς αναπτύχθηκαν πολλές διαδικτυακές υπηρεσίες διανομής όπως το mobile market που κάνουν εύκολη την διαδικασία έκδοσης, διαφήμισης και διανομής τέτοιων προιόντων.

Η διαδικασία ανάπτυξης ηλεκτρονικών παιχνιδιών μπορεί να διαρκέσει απο μερικές μέρες έως και χρόνια και περιλαμβάνει πολλά βήματα. Ξεκινάει απο την σύλληψη μιάς ιδέας, ακολουθεί ένα προσχέδιο που περιέχει το σενάριο, το περιεχόμενο, χαρακτηριστικά, χρονοδιάγραμμα, πιό κοινό στοχεύει και τι απαιτήσεις έχει. Έπειτα ξεκινάει η υλοποίηση που για μεγάλους τίτλους παιχνιδιών αυτό το στάδιο μπορεί να απασχολήσει απο δεκάδες έως εκατοντάδες άτομα με πολλούς διαφορετικούς ρόλους όπως σχεδιαστές, μουσικούς, προγραμματιστές, δοκιμαστές κα. Όταν η υλοποίηση ολοκληρωθεί ή και πολλές φορές πρήν, ξεκινάει η διαδικασία μάρκετινγκ και προώθησης. Αυτή η εργασία ασχολείται καθαρά με το μέρος της υλοποίησης και στοχεύει να κάνει την διαδικασία πιο απλή και γρήγορη.

1.1 Κίνητρο

Κίνητρο για την επιλογή αυτής της εργασίας ήταν το εύρος θεμάτων ενασχόλησης. Η ανάπτυξη ηλεκτρονικών παιχνιδιών περιλαμβάνει υλοποιήσεις πολλών διαφορετικών αλγορίθμων, δομών και αντικειμένων. Μου δόθηκε η ευκαιρία να χρησιμοποιήσω την Java για να αντιμετωπίσω πολλές κατηγορίες προβλημάτων και κατα συνέπεια να αποκτήσω μία καλή βάση για να επεκτείνω την γνώση μου σε αυτή την γλώσσα. Ένα ακόμη κριτήριο για την επιλογή ήταν η επεκτασιμότητα αυτής της εργασίας καθώς δέν ήθελα η ανάπτυξη να τελειώσει με την πτυχιακή. Είναι ένα πρότζεκτ το οποίο μπορεί να δεχθεί συνεχώς επεκτάσεις και βελτιώσεις και στο σημείο που βρίσκετε τώρα είναι ακόμη στην αρχή.

1.2 Σκοπός και στόχοι

Σκοπός αυτής της εργασίας είναι η απλοποίηση της διαδικασίας κατασκευής δισδιάστατων παιχνιδιών εισάγοντας ένα καινούργιο τρόπο περιγραφής και δομής τους που μοιάζει με την παραγωγή συνθετικού βίντεο στο MPEG χρησιμοποιώντας οπτικοακουστικά αντικείμενα (AVOs). Πιό συγκεκριμένα συνθέτει σκηνές απο τέτοια αντικείμενα τα οποία θα πρέπει να είναι παραμετροποιήσιμα και επαναχρησιμοποιήσιμα. Θα μπορούν να αλληλεπιδρούν με το χρήστη και το καθένα θα περιλαμβάνει ρόλους, διαλόγους, κινήσεις και λειτουργίες. Επίσης θα προσφέρει έτοιμες συναρτήσεις και δομές που χρησιμοποιούνται συχνά στην δημιουργία παιχνιδιών όπως για παράδειγμα για εντοπισμό συγκρούσεων, για φόρτωση και διαχείριση εικόνων και ήχων, φωτισμό, είσοδο χρήστη, δικτύωση κα.

Η υλοποίηση αυτής της εργασίας χωρίζετε σε δύο μέρη. Μια βιβλιοθήκη η οποία μπορεί να δομεί σκηνές δίνοντας της εικόνες, ήχους και κείμενο το οποίο θα περιέχει την περιγραφή των αντικειμένων και το σενάριο του παιχνιδιού. Και έναν editor ο οποίος θα βοηθάει τον χρήστη στην δημιουργία του περιεχομένου και την συγγραφή του σεναρίου.

Τελικός στόχος αυτής της εργασίας είναι να δώσει την δυνατότητα στον χρήστη να κατασκευάζει παιχνίδια με τον ελάχιστο δυνατό κώδικα δίνοντας του πολλές έτοιμες λειτουργίες και μια απλή γλώσσα κατανοητή και απο τον ίδιο αλλα και απο την βιβλιοθήκη. Επίσης θα πρέπει να έχει τέτοια δομή ώστε να μήν περιορίζει τον χρήστη στις δυνατότητες της βιβλιοθήκης. Δηλαδή να περιέχει αντικείμενα τα οποία μπορούν να χρησιμοποιηθούν μεμονωμένα και να προσαρμόζονται εύκολα σε κάθε είδος παιχνιδιού.

1.3 Δομή Εργασίας

Μέχρι αυτό το σημείο έγινε αναφορά στο αντικείμενο και τους στόχους αυτής της εργασίας. Το επόμενο κεφάλαιο ασχολείται με την μεθοδολογία υλοποίησης, δηλαδή με όλη την προετοιμασία και προσχεδιασμό του πρότζεκτ. Περιλαμβάνει ανάλυση προβλήματος, συλλογή γνώσης και καθορισμό λειτουργιών.

Το κεφάλαιο τρία είναι το κύριο μέρος και κάθε υποκεφάλαιο ασχολείται με κάθε κατηγορία υλοποίησης αυτής της εργασίας. Σε κάθε κατηγορία γίνετε αναφορά στον τρόπο υλοποίησης, στους αλγόριθμους που χρησιμοποιήθηκαν όπως επίσης και σε άλλους σημαντικούς και στα προβλήματα που εμφανίστηκαν. Επίσης στο τέλος κάθε κεφαλαίου υπάρχουν παραδείγματα χρήσης του κάθε αντικειμένου που αναλύεται με κώδικα απο τα πιο σημαντικά σημεία. Λόγω της έκτασης αυτής της εργασίας και του περιορισμένου χρόνου δέν ήταν εφικτό να γίνει αναλυτική αναφορά στο κώδικα, έτσι το κεφάλαιο τρία χρησιμεύει και ώς οδηγός χρήσης της βιβλιοθήκης.Τέλος το κεφάλαιο τέσσερα αναφέρει τα αποτελέσματα αυτής της εργασίας, σε τι σημείο βρίσκεται και πιό είναι το μέλλον της.

2 Μεθοδολογία υλοποίησης

Καθώς αυτή η εργασία στοχεύει σε σε μία κατηγορία προβλημάτων, δέν υπάρχει μία και μόνο μεθοδολογία για την υλοποίηση της. Η μορφή αυτής της εργασίας μοιάζει πιο πολύ με μια συλλογή αλγορίθμων, δομών και αντικειμένων που χρησιμοποιούνται ώς εργαλεία για την δημιουργία δισδιάστατων παιχνιδιών με κάθε ένα απο αυτά να απαιτεί διαφορετική προσέγγιση. Μία πιό κοινή μεθοδολογία που θα αφορά όλα τα αντικείμενα θα χρησιμοποιηθεί με σκοπό να μπορούμε εύκολα να περιγράψουμε και να δομήσουμε ηλεκτρονικά παιχνίδια απο τέτοια αντικείμενα.

Η υλοποίηση αυτής της εργασίας χωρίζεται σε τρία μεγάλα στάδια. Το πρώτο είναι η κατασκευή μιας βιβλιοθήκης που θα παρέχει αντικείμενα για την δημιουργία δισδιάστατων παιχνιδιών. Θα πρέπει να είναι παραμετροποιήσιμα, φορητά και να προσαρμόζονται εύκολα σε κάθε περίπτωση χρήσης. Επόμενο στάδιο είναι η δημιουργία ενός σέτ εντολών που μπορούν να χρησιμοποιηθούν ως γλώσσα περιγραφής αυτών των αντικειμένων ώστε να αυτοματοποιήσουμε περισσότερο την διαδικασία παραγωγής. Αυτές οι εντολές στην ουσία αντικαθιστούν τον κώδικα που χρειάζεται για το φόρτωμα των πόρων και την δομή τους σε αντικείμενα παιχνιδιού. Τέλος θα κατασκευαστεί ένα γραφικό περιβάλλον που θα μπορεί να συντάσσει τέτοια αρχεία ώστε να κάνει την διαδικασία ακόμη πιο εύκολη. Κάθε ένα απο αυτά τα στάδια χωρίζεται σε πολλές κατηγορίες οι οποίες αναλύονται σε πολλά προβλήματα.

2.1 Ανάλυση Προβλήματος

Αρχή σχεδόν κάθε υλοποίησης είναι η ανάλυση του προβλήματος που ασχολείται. Σε αυτό το στάδιο θέτονται οι κύριοι στόχοι, προσδιορίζονται οι υπηρεσίες που θα προσφέρει και τι κοινό στοχεύει. Μία τεχνική ανάλυσης είναι να διαιρούμε το κύριο πρόβλημα σε μικρότερα και απλούστερα, με την σειρά τους αυτά μπορούν να αναλυθούν σε άλλα ακόμη πιο απλά και η διαδικασία συνεχίζεται έως ότου τα προβλήματα που προκύπτουν τα οποία ως σύνολο εκφράζουν το αρχικό πρόβλημα θεωρούνται αρκετά απλά για αντιμετώπιση.

Πρώτο βήμα είναι να προσδιορίσουμε απο τί αποτελείτε ένα δισδιάστατο παιχνίδι, ποια είναι τα βασικά συστατικά του και ποιά η ελάχιστη πληροφορία που χρειαζόμαστε για να το περιγράψουμε. Πολλά τέτοια ερωτήματα, αλλα και μερικές φορές οι απαντήσεις τους βρίσκονται κατα την υλοποίηση αλλα πάντα είναι καλό και πρέπει να έχουμε ένα προσχέδιο. Μπορούμε να χωρίσουμε τα συστατικά ενός παιχνιδιού σε τρείς κατηγορίες, εικόνα, ήχος και κείμενο. Οπου κείμενο όχι μόνο αυτό που χρησιμοποιείτε ώς περιεχόμενο μέσα στο παιχνίδι, αλλα κείμενο που περιέχει πληροφορία για να περιγράψει το ίδιο το παιχνίδι.

Το επόμενο και μεγαλύτερο βήμα της ανάλυσης είναι να συλλέξουμε όλα τα αντικείμενα και τους αλγόριθμους που χρησιμοποιούνται συχνά στην υλοποίηση παιχνιδιών και να τους δώσουμε τέτοια δομή ώστε να μπορούν να δουλέψουν μαζί, ανεξάρτητα αλλά και σε συνδυασμό με ξένο κώδικα. Να έχουν όσο το δυνατόν μία γενική αλλά συγχρόνως ολοκληρωμένη περιγραφή. Κυριότερες κατηγορίες αυτού του βήματος είναι:

- Rendering – Animation – Lighting
- Ήχος
- Είσοδος Χρήστη
- Συστήματα Σωματιδίων
- Συγκρούσεις
- Δικτύωση
- Σκηνές

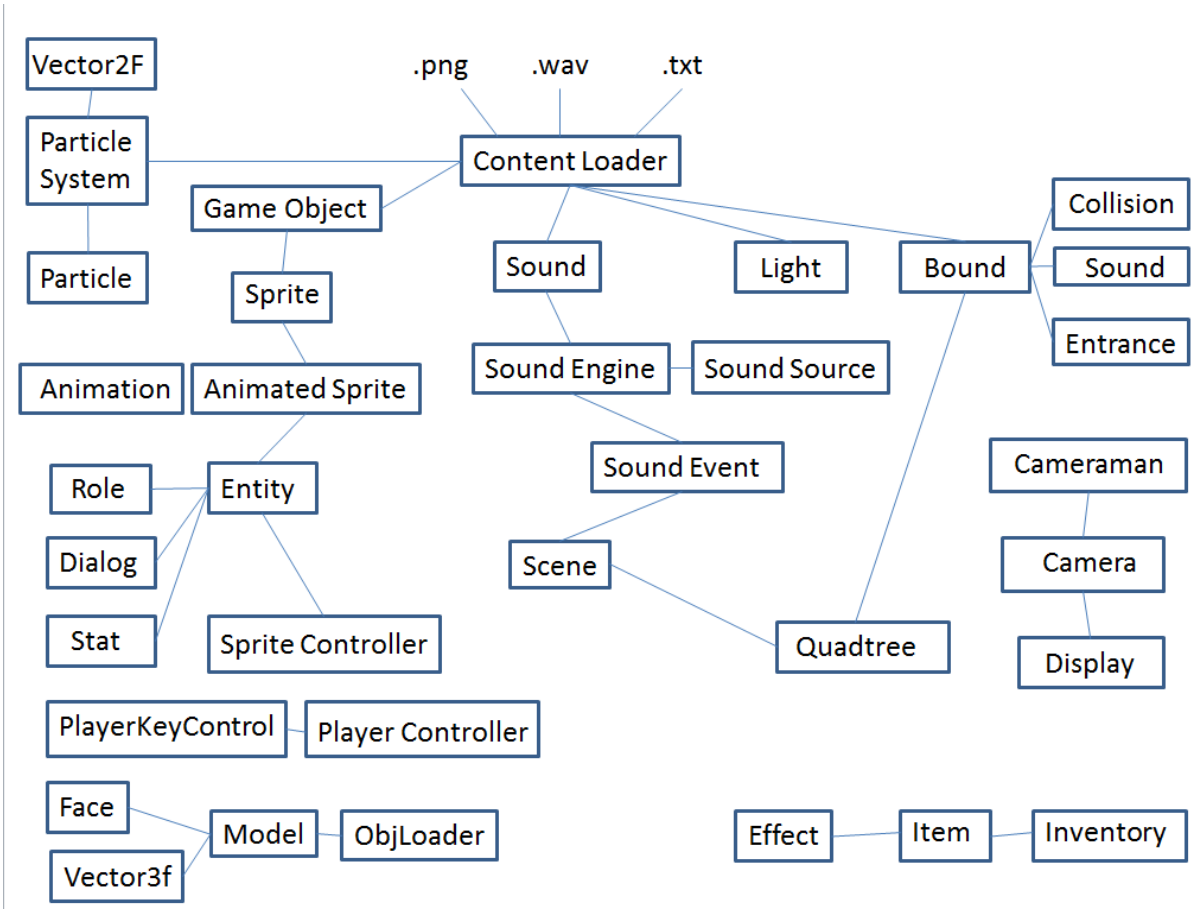
2.2 Συλλογή Πληροφοριών

Εδώ η απάντηση είναι μία, το Ίντερνετ. Η δυνατότητα που έχουμε πλέον να ψηφιοποιούμε και να μεταδίδουμε οποιοδήποτε είδους πληροφορία με τόση ευκολία οδήγησε στην δημιουργία ενός πελώριου όγκου δεδομένων προσβάσιμο σχεδόν σε όλους. Στο ίντερνετ μπορεί κανείς να βρεί οτιδήποτε αρκεί να ξέρει πως να ψάξει. Απο παραδείγματα για υλοποιήσεις αλγορίθμων, τεχνικές προσέγγισης προβλημάτων, best-practices προγραμματισμού μέχρι και μαθήματα ζωγραφικής και επεξεργασίας ήχου.

Μία σημαντική πηγή για αυτήν την εργασία ήταν το βιβλίο Killer Game Programming In Java του Andrew Davison. Αναφέρετε σε πολλά θέματα προγραμματισμού παιχνιδιών και περιέχει πολλούς αλγόριθμους και παραδείγματα.

2.3 Σχέδιο Υλοποίησης

Σε αυτό κεφάλαιο γίνετε μια σύντομη αναφορά στην δομή του JIGL και του GCE και των αντικειμένων που αποτελούνται. Εδώ παρουσιάζονται περιληπτικά τα πιό βασικά αντικείμενα άν και η εργασία περιέχει πολλά επιπλέον, λόγω χρόνου δέν αναφέρονται εδώ αλλά στα επόμενα κεφάλαια. Η εικόνα πιο κάτω παρουσιάζει τα βασικότερα αντικείμενα που χρησιμοποιεί η βιβλιοθήκη και πώς αυτά συνδυάζονται μεταξύ τους.



Εικόνα 1 Δομή JIGL

Content Loader

Το αντικείμενο ContentLoader είναι υπεύθυνο για την μετατροπή εικόνας, ήχου και κειμένου σε αντικείμενα τα οποία μπορεί να χρησιμοποιήσει η βιβλιοθήκη ή και μεμονωμένα ο χρήστης. Άν και αυτός είναι ο μοναδικός σκοπός του και δέν συνδυάζετε με τα υπόλοιπα αντικείμενα είναι απο τα σημαντικότερα μέρη της υλοποίησης καθώς μειώνει σημαντικά τον κώδικα που απαιτείτε από τον χρήστη.

Display – Camera

Το αντικείμενο Display κληρονομεί απο το JPanel της Java και είναι η επιφάνεια στην οποία ζωγραφίζετε η τελική εικόνα. Ενώ τα αντικείμενα Camera και Cameraman ορίζουν που "κοιτάει" ο παίκτης μέσα στην σκηνή και πρέπει να συνδυάζονται και με το Display αλλά και με την σκηνή ώστε να υπολογίζουμε το viewport και να εφαρμόσουμε το κατάλληλο culling.

Game Object

Καθώς όλα τα γραφικά αντικείμενα των ηλεκτρονικών παιχνιδιών έχουν κάποια κοινά στοιχεία όπως θέση, διαστάσεις, αναγνωριστικά και όνομα, πρέπει να υπάρχει μια βάση απο την οποία όλα αυτά τα αντικείμενα θα κληρονομούν αυτά τα κοινά χαρακτηριστικά. Έπειτα κάθε αντικείμενο που χρειάζεται μια πιό ολοκληρωμένη υλοποίηση, θα κληρονομεί απο αυτήν βάση και θα προσφέρει επιπλέον χαρακτηριστικά στην ίδια του την κλάση. Για παράδειγμα το αντικείμενο Sprite είναι τύπου GameObject αλλα επεκτείνει την λειτουργικότητα του ώστε να περιέχει μια εικόνα και μια συνάρτηση για το πώς αυτό το αντικείμενο θα ζωγραφιστεί. Με την σειρά του το αντικείμενο AnimatedSprite επεκτείνει την κλάση Sprite ώστε να περιέχει animations. Το ίδιο για το αντικείμενο Entity το οποίο θα μπορεί να περιέχει ρόλους, διαλόγους και στατιστικά. Αυτή η δομή χρησιμεύει ώστε να διαφοροποιήσουμε την λειτουργικότητα ανάμεσα στα αντικείμενα. Για παράδειγμα όλα τα παραπάνω περιέχουν μια συνάρτηση (draw()) που ορίζει πώς αυτά τα αντικείμενα θα ζωγραφιστούν στην οθόνη και την οποία κληρονομούν απο την κλάση Sprite. Αλλιώς όμως ζωγραφίζονται οι στατικές εικόνες, αλλιώς τα animations και αλλιώς τα συστήματα σωματιδίων. Με αυτόν τον τρόπο δέν πετυχαίνουμε μόνο καλύτερη οργάνωση στον κώδικα αλλά επιτρέπουμε στον χρήστη να επιλέξει την πολυπλοκότητα των αντικειμένων που θα χρησιμοποιήσει απο μια ιεραρχική δομή. Για παράδειγμα δέν έχει νόημα να χρησιμοποιήσουμε αντικείμενα με animation αν χρειάζετε να ζωγραφίσουμε μόνο μία εικόνα. Επίσης με αυτόν τον τρόπο μπορούμε να συνδυάσουμε διαφορετικά αντικείμενα σε μία κοινή δομή και να τα αντιμετωπίσουμε ώς ένα. Παράδειγμα άν θέλουμε να ζωγραφίσουμε όλα τα αντικείμενα, δέν χρειάζεται να ξέρουμε αν το αντικείμενο είναι Sprite ή Particle System εφόσον ξέρουμε ότι όλα περιέχουν την συνάρτηση draw().

Particle System

Τα συστήματα σωματιδίων χρησιμοποιούν το αντικείμενο Vector2f το οποίο αναπαριστά ένα διάνυσμα και χρησιμεύει για την θέση, ταχύτητα, επιτάχυνση ακόμα και για τις δυνάμεις που εφαρμόζουμε στο σύστημα. Είναι μία απλή δομή δύο τιμών float. Το αντικείμενο Particle περιέχει όλα τα χαρακτηριστικά με τα οποία το εκκινεί το σύστημα. Είναι η "υπόσταση" κάθε σωματιδίου που δημιουργούμε με τα δικά του μοναδικά χαρακτηριστικά. Τέλος το Particle System αναπαρηστά όλο το σύστημα το οποίο δημιουργεί, ενημερώνει, ζωγραφίζει και καταστρέφει τα σωματίδια.

Bounds – Quadtree

Τα bounds είναι σχήματα τα οποία ορίζουν περιοχές που εκκινούν συγκρούσεις, ενώ το Quadtree είναι ο αλγόριθμος για τον εντοπισμό αυτών. Και τα δύο χρησιμοποιούνται απο την σκηνή η οποία μπορεί να πυροδοτεί Events. Αυτές οι περιοχές καθώς και τα event που εκκινούν χωρίζονται σε τρείς κατηγορίες. Συγκρούσεων, έναρξης ήχων και εναλλαγής σκηνών. Για τις δύο τελευταίες η βιβλιοθήκη μπορεί να διαχειριστεί μόνη της αυτά τα γεγονότα. Για την πρώτη όμως είναι υπεύθυνη μόνο για τον εντοπισμό.

SoundEngine

Είναι το αντικείμενο που διαχειρίζεται τους ήχους και μπορεί να δεχτεί event είτε απο τον χρήστη είτε απο την σκηνή. Κρατάει όλους τους ήχους που έχουμε φορτώσει και εκκινεί τον κάθε ένα σε ξεχωριστό νήμα. Έχει επιλογές έντασης και τρόπου αναπαραγωγής.

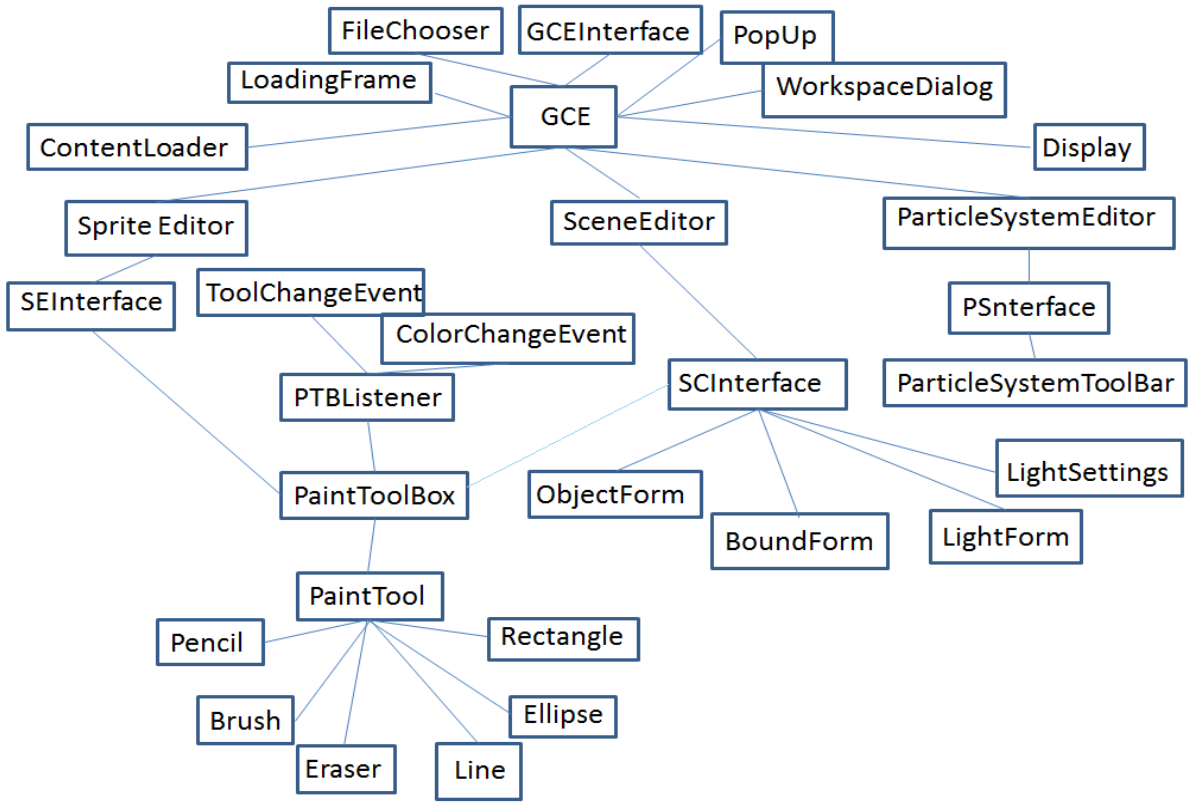
Sprite Controller

Αυτή η κλάση δέχεται αντικείμενα τύπου Entity και χρησιμοποιήστε για να αυτοματοποιήσουμε την συμπεριφορά αυτών των αντικειμένων ως προς την κίνηση, animation και ήχου. Αυτή η συμπεριφορά περιγράφετε απο τα αντικείμενα Role που αυτές οι κλάσεις περιέχουν.

ObjLoader

Αντικείμενο το οποίο μπορεί να φορτώσει τρισδιάστατα μοντέλα απο obj αρχεία. Χρησιμοποιεί τις κλάσεις Vector3f και Face για τα περιγράψει. Είναι το πρώτο βήμα για την εισαγωγή της βιβλιοθήκης στο 3D και δέν συνδυάζετε ακόμη με άλλα αντικείμενα αυτής.

Το δεύτερο μέρος της εργασίας που είναι ο Editor έχει την παρακάτω δομή.



Εικόνα 2 Δομή GCE

GCE

Αυτό το αντικείμενο είναι υπεύθυνο για τον έλεγχο του workspace, την εκκίνηση του προγράμματος και όλων των editor. Χρησιμοποιεί το GCEInterface ώς γραφικό περιβάλλον. Οι κλάσεις WorkspaceDialog, PopUp, FileChooser και LoadingFrame είναι βοηθητικά αντικείμενα που χρησιμοποιούνται απο όλους τους editor. Τα αντικείμενα ContentLoader και Display είναι τα ίδια της βιβλιοθήκης.

ParticleSystemEditor

Αυτό το αντικείμενο διαχειρίζεται το σύστημα σωματιδίων που κάνουμε edit. Περιέχει όλες τις συναρτήσεις για να σώσουμε, να φορτώσουμε και να επεξεργαστούμε συστήματα. Το PSInterface είναι η γραφική διεπαφή που αποτελείτε απο ένα Display και το ParticleSystemToolbar το οποίο περιέχει όλες τις επιλογές παραμετροποίησης του συστήματος.

SpriteEditor

Ο Sprite Editor διαχειρίζεται τα Sprite, animated και μή. Είναι ένα εργαλείο ζωγραφικής το οποίο έχει την δυνατότητα να δομεί και να αναπαράγει animation. Το SEInterface είναι η γραφική διεπαφή η οποία χρησιμοποιεί το PaintToolBox.

SceneEditor

Ο Scene Editor περιέχει την λειτουργικότητα για να δημιουργούμε και να διαχειριζόμαστε σκηνές. Το SCInterface είναι η γραφική διεπαφή η οποία χρησιμοποιεί κάποιες επιπλέον φόρμες (ObjectForm, BoundForm, LightForm, LightSettings) για να διαχειρίζεται τα αντικείμενα της σκηνής. Αυτό το αντικείμενο όπως και τα δύο προηγούμενα εξάγουν το αποτέλεσμα σε μία μορφή κατανοητή από την βιβλιοθήκη, έτσι ότι δημιουργούμε μπορούμε να το χρησιμοποιήσουμε άμεσα με αυτήν.

PaintToolBox

Το PaintToolBox είναι η εργαλειοθήκη που διαχειρίζεται τα αντικείμενα Pencil, Brush, Eraser, Line, Rectangle, Ellipse τα οποία χρησιμοποιούμε για να ζωγραφίσουμε. Λειτουργεί σε συνδυασμό με το PTBListener ο οποίος είναι ακροατής συμβάντων που πυροδοτεί τα ToolChangeEvent και ColorChangeEvent. Αυτή η εργαλειοθήκη χρησιμοποιείτε και με τον SpriteEditor και με τον SceneEditor.

(Davison)

3 Κύριο Μέρος

Σε αυτό το κεφάλαιο θα αναλυθούν τα σημαντικότερα μέρη της υλοποίησης, θα γίνει αναφορά σε αλγόριθμους που χρησιμοποιήθηκαν και μή, στα προβλήματα που προέκυψαν και τις λύσεις τους. Επίσης στο τέλος κάθε υποκεφάλαιου υπάρχουν παραδείγματα χρήσης του αντικειμένου που ασχολείτε, με κομμάτια σχολιασμένου κώδικα και παραδείγματα δομής αρχείων.

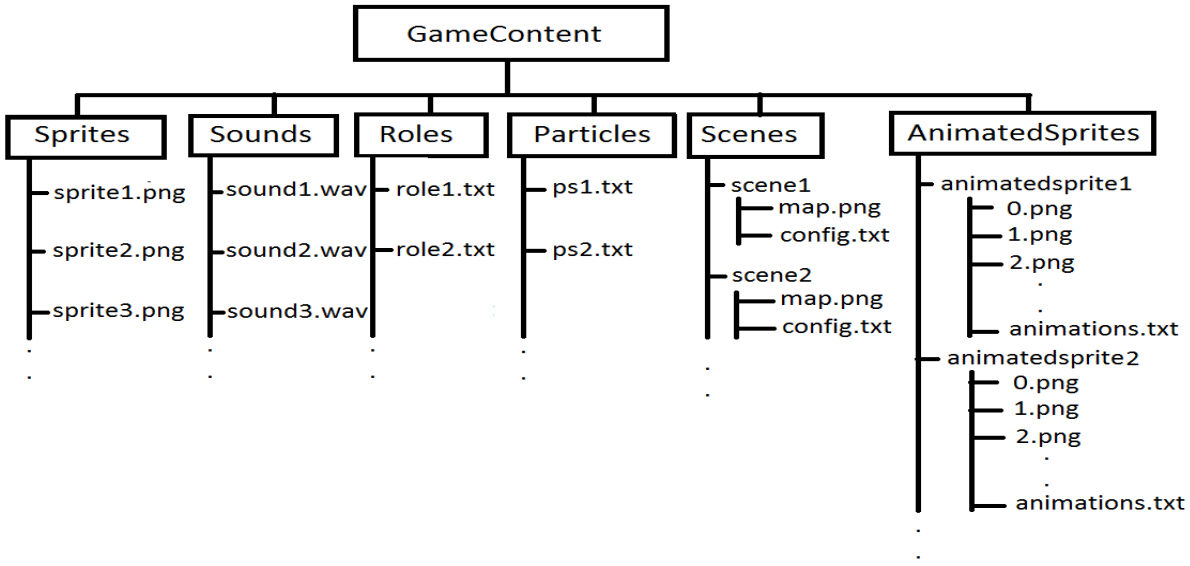
3.1 Διαχείριση και Δομή Αντικειμένων

Αυτό το κεφάλαιο είναι σημαντικό για τον αναγνώστη καθώς εξηγεί πώς αναπαριστώνται και δομούνται πολλά απο τα αντικείμενα που αναφέρονται στα επόμενα κεφάλαια, απο τί αρχεία αποτελούνται και πού πρέπει να βρίσκονται στο δέντρο φακέλων που χρησιμοποιεί η βιβλιοθήκη. Επίσης σε αυτό το κεφάλαιο θα αναλυθεί η γλώσσα με την οποία περιγράφονται τα αντικείμενα, πράγμα σημαντικό για την κατανόηση των επόμενων κεφάλαιων.

3.1.1 Το Δέντρο Αρχείων του GCE

Ξεκινώντας απο τα βασικά, το δέντρο αρχείων που περιέχει όλο το περιεχόμενο του παιχνιδιού πρέπει να είναι όπως της εικόνας πιο κάτω. Ο αρχικός φάκελος (GameContent) είναι η διαδρομή αρχείου που δίνουμε στην βιβλιοθήκη για να μπορεί να βρεί όλα τα αντικείμενα, έτσι πρέπει να έχει συγκεκριμένη δομή.

- Sprites και Sounds: περιέχουν απλές εικόνες (png) για στατικά sprite και ήχους (wav) για μουσική και εφέ αντίστοιχα.
- Roles: αρχεία κειμένου τα οποία περιέχουν εντολές κίνησης και αλλαγής animation, χρησιμοποιούνται ως ρόλοι για αντικείμενα που δεν ελένχονται απο τον παίχτη.
- Particles: αρχεία κειμένου που το καθένα περιέχει όλες της μεταβλητές που καθορίζουν την φύση και την συμπεριφορά ενός συστήματος σωματιδίων.
- AnimatedSprites: ξεχωριστός φάκελος για κάθε sprite. Περιέχει μια συλλογή εικόνων (png) αριθμημένα πάντα απο μηδέν έως το πλήθος των καρέ και ένα αρχείο κειμένου (animations.txt) που περιέχει όλα τα animation του αντικειμένου με τα καρέ αυτά.
- Scenes: ξεχωριστός φάκελος για κάθε scene. Περιέχει μια εικόνα (map.png) που χρησιμοποιείτε ως background για την σκηνή και ένα αρχείο κειμένου (config.txt) με διάφορα χαρακτηριστικά της καθώς και όλα τα αντικείμενα που περιέχει τα οποία μπορεί να είναι όλα τα παραπάνω.



Εικόνα 3 Δέντρο Αρχείων GameContent

Κάθε αρχείο κειμένου περιέχει εντολές για την βιβλιοθήκη που ορίζουν τα αντικείμενα καθώς και σχόλια για καθοδήγηση. Αυτά τα αρχεία μπορούν είτε να παραχθούν απο τους editor του GCE και να τροποποιηθούν μετά από τον χρήστη ή να γραφτούν εξολοκλήρου από αυτόν. Πιο συγκεκριμένη αναφορά για την σύνταξη αυτών των αρχείων γίνετε σε επόμενα κεφάλαια που ασχολούνται με το κάθε αντικείμενο.

3.1.2 Το αντικείμενο ContentLoader

Το πρώτο και ίσως το σημαντικότερο αντικείμενο της βιβλιοθήκης που θα αναφερθεί. Είναι το αντικείμενο που διαβάζει όλα τα αρχεία κειμένου, εικόνες και ήχους και τα μεταφράζει σε αντικείμενα παιχνιδιού. Μας δίνει την δυνατότητα να φορτώσουμε απο απλές εικόνες μέχρι ολοκληρωμένες σκηνές με την χρήση ελάχιστου κώδικα. Δέν μπορεί να γίνει πολύ ανάλυση σε αυτό το αντικείμενο καθώς αυτός είναι ο μοναδικός σκοπός του, δεν χρησιμοποιεί ειδικούς αλγόριθμους ούτε έχει κάποια αξιοσημείωτη δομή αλλά είναι σημαντικό γιατί είναι το αντικείμενο που γλιτώνει τον χρήστη απο τον περισσότερο κόπο.

Δίνοντας του την διαδρομή για το αρχείο GameContent (που αναφέρθηκε στο προηγούμενο κεφάλαιο) μπορούμε να φορτώσουμε οποιοδήποτε αντικείμενο που περιγράφετε στην γλώσσα που χρησιμοποιεί η βιβλιοθήκη. Αυτό το αντικείμενο είναι απλά μια μεγάλη συλλογή συναρτήσεων κάθε μία εξειδικευμένη να διαβάζει συγκεκριμένο αρχείο και να επιστρέφει από αντικείμενα παιχνιδιού τα οποία εύκολα προσθέτουμε στην σκηνή μέχρι και ολόκληρες σκηνές με πολλά αντικείμενα μέσα που μπορούμε να προσπελάσουμε.

Η μοναδική επιπλέον λειτουργία είναι να αποδίδει αυτόματα αναγνωριστικά όταν φορτώνουμε αντικείμενα με το ίδιο όνομα. Για παράδειγμα μπορεί να έχουμε ένα sprite που να απεικονίζει ένα δέντρο, αυτό το sprite μπορεί να υπάρχει πάνω απο μία φορά σε κάθε σκηνή καθώς δεν χρησιμοποιείτε ποτέ μοναδική εικόνα σε αντικείμενα που επαναλαμβάνονται. Έτσι έχουμε την δυνατότητα να φορτώσουμε αυτόματα μια σκηνή αλλά και να μπορούμε να προσπελάσουμε τέτοια αντικείμενα και να τα χειριζόμαστε μεμονωμένα.

Παραδείγμα χρήσης ContentLoader:

```
ConttentLoader cL = new ContentLoader("C:/GCE/GameContent");

////Φόρτωση σκηνής:

Scene scene1 = contentLoader.loadSceneByName(sceneName);

///Αν θέλουμε να πάρουμε ένα αντικείμενο απο την σκηνή:

AnimatedSprite player = (AnimatedSprite) mainScene.getSprite(id, layer);

///id: αναγνωρηστικό αντικειμένου,  layer: επίπεδο της σκηνής (άν είναι γνωστό για πιο γρήγορη
///αναζήτηση)
///ή απλά:

AnimatedSprite player = (AnimatedSprite) mainScene.getSprite(id);

///Φόρτωση ξεχωριστών αντικειμένων:

Sprite sprite = contentLoader.loadSprite(spriteId);
AnimatedSprite aSprite = contentLoader.loadAnimatedSprite(aSpriteId);
Role role = contentLoader.loadRole(roleId);
ParticleSystem ps = contentLoader.loadParticleSystem(id);
```

3.2 Είσοδος Χρήστη

Η Java ως γλώσσα υψηλού επιπέδου παρέχει πολλές έτοιμες υλοποιήσεις για είσοδο χρήστη, όμως δεν καλύβει πολλές απο τις απαιτήσεις κάποιων έως των περισσότερων ηλεκτρονικών παιχνιδιών που χρειάζονται γρήγορους και σύνθετους χειρισμούς. Χειρισμούς που εξαρτώνται πολύ από τον χρόνο και από πολλές εισόδους. Συνδυασμός πληκτρολογίου με ποντικιού, πολλαπλά πλήκτρα, είσοδοι με χρονικά περιθώρια και διατήρηση σωστών συντεταγμένων ανεξάρτητα της θέασης (τοποθεσία, ζουμ) είναι μερικά παραδείγματα. Επίσης ο κώδικας για υλοποίηση αντικειμένων που χειρίζονται την είσοδο χρήστη δεν είναι τόσο περίπλοκος αλλά είναι είναι μεγάλος σε έκταση πράγμα που κάνει την οργάνωση πιο δύσκολη. Η βιβλιοθήκη έχει δύο αντικείμενα που βοηθάνε σε τέτοιες υλοποιήσεις τα οποία παρέχουν κάποια επιπλέον χαρακτηριστικά από τα πρότυπα της Java και είναι πιο εύκολα στην χρήση.

Το πρώτο αντικείμενο για είσοδο πληκτρολογίου, PlayerKeyControl είναι μια abstract κλάση η οποία δηλώνει δύο συναρτήσεις που πρέπει να οριστούν από τον δημιουργό του αντικειμένου. Η μία συνάρτηση είναι για πάτημα πλήκτρου και η άλλη για ελευθέρωση. Αφού φτιάξουμε τέτοια controls τα προσθέτουμε στον PlayerController και τα αντιστοιχούμε σε πλήκτρα. Με αυτόν τον τρόπο ο χρήστης μπορεί να ορίσει συμπεριφορές για αυτά τα γεγονότα απλά συμπληρώνοντας τις μεθόδους που προκύπτουν στην αρχικοποίηση αυτών των αντικειμένων με τον δικό του κώδικα.

Το δεύτερο αντικείμενο PlayerController χειρίζεται τα PlayerKeyControl καθώς επίσης και την είσοδο ποντικιού. Αφού έχουμε φτιάξει αντικείμενα PlayerKeyControl τα δηλώνουμε στο PlayerController το οποίο τρέχει σε δικό του νήμα και εκτελεί όλα τα control που σχετίζονται με γεγονότα εισόδου. Αυτό το αντικείμενο κάνει implement τις κλάσεις ακροατών της Java οπότε θα πρέπει να προστεθεί στο αντικείμενο που θέλουμε να κάνει focus με τον ίδιο τρόπο που προσθέτουμε τους πρότυπους της Java και είναι abstract που σημαίνει οτι ο χρήστης μπορεί να επαναπροσδιορίσει της πρότυπες συναρτήσεις της Java αυτών των ακροατών όπως MouseClicked, MouseMoved κλπ. Αυτό το αντικείμενο χρησιμοποιεί ίδη τις μεθόδους των ακροατών για να μπορούμε εύκολα να πάρουμε την θέση του ποντικιού ή αν κάποιο απο τα κουμπιά του είναι πατημένο αλλα δίνουμε την ελευθερία στον χρήστη να τις επαναπροσδιορίσει για να καλύψει τις ανάγκες του. Εκτός από κάποιες επιπλέον βασικές συναρτήσεις του PlayerController μία σημαντική λειτουργία είναι ο click-buffer ο οποίος αποθηκεύει τους χρόνους των τελευταίων ‘x’ κλίκ, όπου ‘x’ αριθμός που ορίζει ο χρήστης.

Μία μελλοντική εξέλιξη αυτού του αντικειμένου θα είναι η δυνατότητα να ορίσουμε χειρισμούς με πολλαπλές εισόδους. Όπως για παράδειγμα λειτουργίες που απαιτούν πολλαπλά πλήκτρα ή συνδυασμό πληκτρολογίου – ποντικιού ή ακόμη και μεταβλητές του παιχνιδιού. Αυτό είναι ίδη δυνατό αλλά μπορεί να αυτοματοποιηθεί ακόμη περισσότερο ώστε να έχει ο χρήστης μία πιο οργανωμένη και καθαρή υλοποίηση.

Παραδείγμα χρήσης PlayerKeyControl:

```
PlayerKeyControl pc1 = new PlayerKeyControl() {     //Μόλης φτιάχνουμε ένα control μας ζητάει να
                                                      ορίσουμε δύο συναρτήσεις.
        @Override
        public void onPressControl() {                  //Πάτημα του κουμπιού
          /////// User code example:
          /////// player.setActiveAnimation("walk");
          /////// soundEvent(new SoundEvent("steps.wav"));
          //////  player.move(0, -1);
        }
        @Override
        public void onReleaseControl() {                // Ελευθέρωση του κουμπιού
           /////// User code example:
           ///////player.setActiveAnimation("idle");
```

```
        }

PlayerKeyControl pc2 = new PlayerKeyControl() {

        @Override
        public void onPressControl() {
          /////// User code example:
          /////// player.setActiveAnimation("walk");
          /////// soundEvent(new SoundEvent("steps.wav"));
          //////  player.move(0, 1);
        }

        @Override
        public void onReleaseControl() {
           /////// User code example:
           ///////player.setActiveAnimation("idle");
        }
      };
```

Παραδείγμα χρήσης PlayerController:

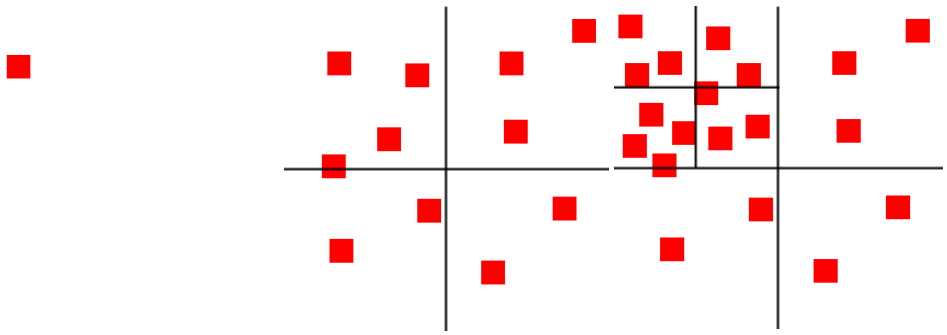
```
PlayerController controler = new PlayerController() { };  //Δημιουργούμε τον controller

controler.addControll(pc1, KeyEvent.VK_W);          // Προσθέτουμε τα KeyControl
controler.addControll(pc2, KeyEvent.VK_S);          // pc1,pc2 στον controller
                                                    // και τα αντιστοιχούμε στα κουμπιά
                                                    //  ¨W¨ και ¨S¨ .

frame.addKeyListener(controler);                    // Ορίζουμε τον controller ώς ακροατή στο
                                                    // frame του παιχνιδιού.

controler.start();                                  // Εκκινούμε το νήμα του controller.
```

3.3 Συγκρούσεις

Οι συγκρούσεις[1] είναι συμβάντα μέσα στο παιχνίδι που ορίζουν πότε δύο αντικείμενα ¨ακουμπάνε¨. Κάθε αντικείμενο ορίζει ένα χώρο που καταλαμβάνει μέσα στην σκηνή ως σχήμα (τετράγωνο, κύκλος) το οποίο λέγετε bound και όταν ψάχνουμε για συγκρούσεις, ελέγχουμε αν κάποια απο τις συντεταγμένες των σχημάτων είναι κοινές. Είναι μια βαριά διεργασία όταν έχουμε να εκλέξουμε πολλά αντικείμενα που έχουν συνήθως τα παιχνίδια.

3.3.1 Quadtrees

Η βιβλιοθήκη για την αναζήτηση συγκρούσεων χρησιμοποιεί την τεχνική Quadtree[2]. Το quadtree είναι μια δενδροειδής δομή που διαιρεί τον χώρο σε τέσσερις κόμβους. Όταν προσθέτουμε αντικείμενα στο δέντρο, τοποθετούνται στους κόμβους ανάλογα την θέση τους και οταν φτάσουμε ένα όριο, ο κόμβος διαιρείτε πάλι σε τέσσερις υπο-κόμβους. Όταν θέλουμε να ελέγξουμε για συγκρούσεις ζητάμε απο το quadtree όλα τα αντικείμενα που βρίσκονται στον ίδιο κόμβο με αυτό που θέλουμε να εκλέξουμε, δηλαδή όσα είναι ‘κοντά’. Με αυτόν τον τρόπο αποφεύγουμε να ελέγχουμε όλα τα αντικείμενα της σκηνής και ελέγχουμε μόνο όσα είναι πιθανά να συγκρουστούν.



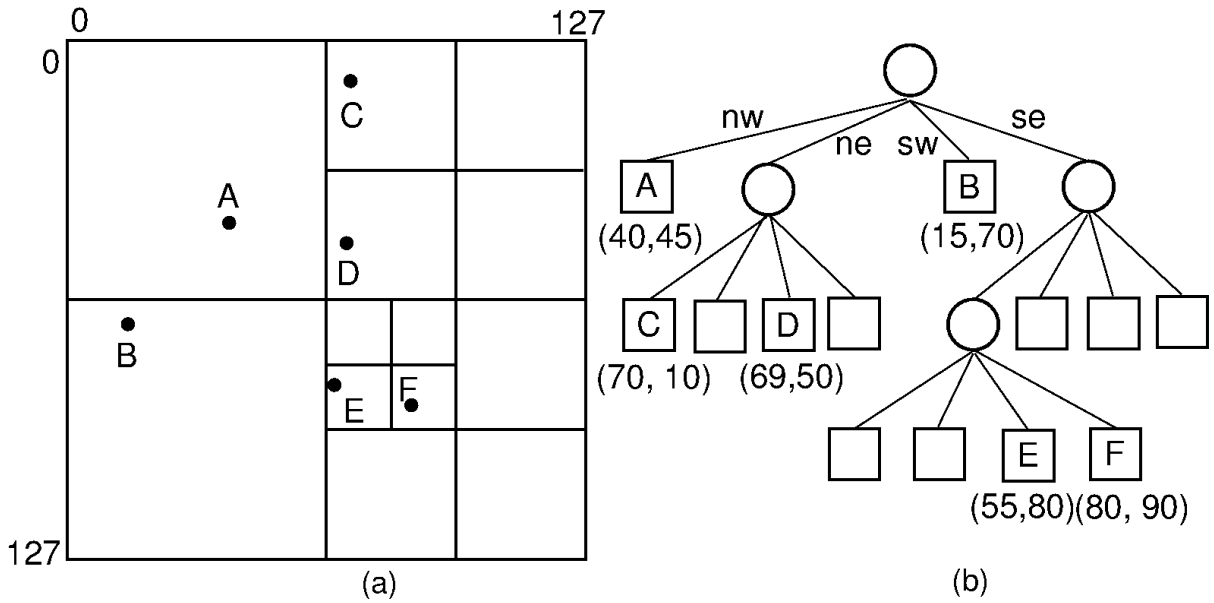
Εικόνα 4 Λειτουργία Quadtree

Στην εικόνα από πάνω βλέπουμε ότι όσο προσθέτουμε αντικείμενα σε μια περιοχή τόσο αυτή διαιρείται ώστε να μειώσουμε την επιφάνεια αναζήτησης. Έτσι όταν θέλουμε για παράδειγμα να ελέγξουμε το αντικείμενο τέρμα πάνω-αριστερά στην κατάσταση τρία της εικόνας «γ» , ελέγχουμε μόνο με όσα βρίσκονται στον ίδιο κόμβο. Σε αυτήν την περίπτωση μόνο με δύο από όλα τα αντικείμενα του δέντρου. Το αντικείμενο Quadtree της βιβλιοθήκης περιέχει έτοιμες συναρτήσεις που κάνουν την χρήση του πιο εύκολη. Επιλέγει μόνο του τον σωστό κόμβο κάθε φορά που εισάγουμε

[1] http://devmag.org.za/2009/04/13/basic-collision-detection-in-2d-part-1/
[2] http://en.wikipedia.org/wiki/Quadtree

κάποιο αντικείμενο και βρίσκει και επιστρέφει όλα τα αντικείμενα κοντά σε κάθε ένα που θέλουμε να εκλέξουμε.

Αυτή η τεχνική δεν χρησιμοποιείτε μόνο στις συγκρούσεις αλλά και σε πολλές άλλες εφαρμογές όπως ανάλυση μορφοκλασματικών εικόνων, οπτικό πεδίο σε κάμερες, spatial indexing, αποθήκευση δεδομένων, μορφολογία πεδίου κα.



Εικόνα 5 Δέντρο Quadtree

3.3.2 Συμβάντα Ήχων και Εναλλαγής Σκηνών

Συγκρούσεις μπορούν να χρησιμοποιηθούν και για άλλες υλοποιήσεις εκτός από τον περιορισμό κίνησης. Μπορούμε να αξιοποιήσουμε τα γεγονότα συγκρούσεων με πολλούς τρόπους όπως για παράδειγμα εκκίνηση ήχων, είσοδο σε περιοχή, εναλλαγή σκηνών. Τα CollisionBound που έχει η βιβλιοθήκη ως bounds συγκρούσεων χωρίζονται σε τρεις τύπους:

Collision: Χρησιμοποιούνται ως απλά bound συγκρούσεων.
Entrance: Εναλλαγή σκηνών, συμβάντα εισόδου σε περιοχή.
Sound: Εκκινούν ήχους όταν ο παίκτης μπει σε μια περιοχή.

Μπορούμε να προσθέσουμε τέτοια bounds σε μια σκηνή είτε μέσω του Scene Editor είτε γράφοντας τις δηλώσεις στο config.txt της. Ο τρόπος σύνταξης τέτοιων bound για να τα δηλώσουμε κατευθείαν μέσα στο αρχείο της σκηνής και να φορτωθούν αυτόματα είναι ο παρακάτω.

Για απλά collision:

CBCR:x/y/w/h

Για bound ήχου:

CBSR:soundId/mode/volume#x/y/w/h

Για εναλλαγή σκηνής:

CBER:sceneId#1/300/1000/400

Αρχή κάθε δήλωσης ξεκινάει με “CB” και ακολουθεί ο τύπος σύγκρουσης (C: collision,S: sound, E: entrance) και το σχήμα του bound (R: rectangle , E: ellipse). Έτσι για παράδειγμα άν θέλουμε ένα κύκλο που εκκινεί ήχο θα δηλώσουμε: CBSE. Μετά απο αυτήν την δήλωση ακολουθούν τα χαρακτηριστικά. Εκτός απο τον τύπο collision που δηλώνει μόνο θέση και διαστάσεις (x,y,w,h), τα υπόλοιπα έχουν κάποιες επιπλέον τιμές. Για ήχο η τιμή soundID είναι το όνομα του αρχείου ήχου μαζί με την κατάληξη (seaB.wav) , η τιμή mode είναι ή ‘p’ ή ‘l’ για play και loop αντίστοιχα που ορίζει άν ο ήχος παίξει μία φορά ή συνεχόμενα. Το volume είναι η ένταση που επιθυμούμε να αναπαραχθεί ο ήχος. Για bounds τύπου entrance η τιμή sceneId είναι το όνομα της σκηνής που θέλουμε να μεταβεί ο παίχτης. Άν θέλουμε μπορούμε να δημιουργήσουμε τέτοια bounds και μέσα απο κώδικα όπως στα παραδείγματα πιο κάτω.

Παραδείγμα χρήσης Quadtree:

QuadTree quadTree = new QuadTree(level,rectangle);
///level: αρχικό επίπεδο, rectangle: bound του χώρου.

///προσθήκη αντικειμένων:

quadTree.insert(objBounds); //*objBounds: το bound σχήμα του αντικειμένου.*

//ανάκτηση όλων των κοντινών αντικειμένων:

quadTree.retreive(found, bound);

//found: Vector που επιστρέφονται τα αποτελέσματα, χρησιμοποιείτε γιατί η συναρτηση είναι αναδρομική.
//bound: Το σχήμα bound που θέλουμε να ελένξουμε.

//καθάρισμα δέντρου:

quadTree.clear();

Η βιβλιοθήκη μπορεί να διαχειριστεί αυτόματα συμβάντα έναρξης ήχων και εναλλαγής σκηνών. Όμως για συμβάντα συγκρούσεων είναι υπεύθυνη μόνο για τον εντοπισμό τους. Το πώς θα αξιοποιηθούν είναι στο χέρι του χρήστη. Επίσης η σκηνή περιέχει από μόνη της ένα αντικείμενο Quadtree το οποίο χρησιμοποιεί μόνο για τα ανεξάρτητα bound της σκηνής που ορίζει ο χρήστης και όχι για τα bound που περιέχουν τα υπόλοιπα αντικείμενα. Δηλαδή μόνο αυτά που περιγράφουν την σκηνή όπως οι είσοδοι/έξοδοι, περιοχές με ήχο και απρόσβατα σημεία. Για συγκρούσεις μεταξύ αντικειμένων όπως Sprites ο χρήστης μπορεί να χρησιμοποιήσει ξεχωριστό αντικείμενο Quadtree.

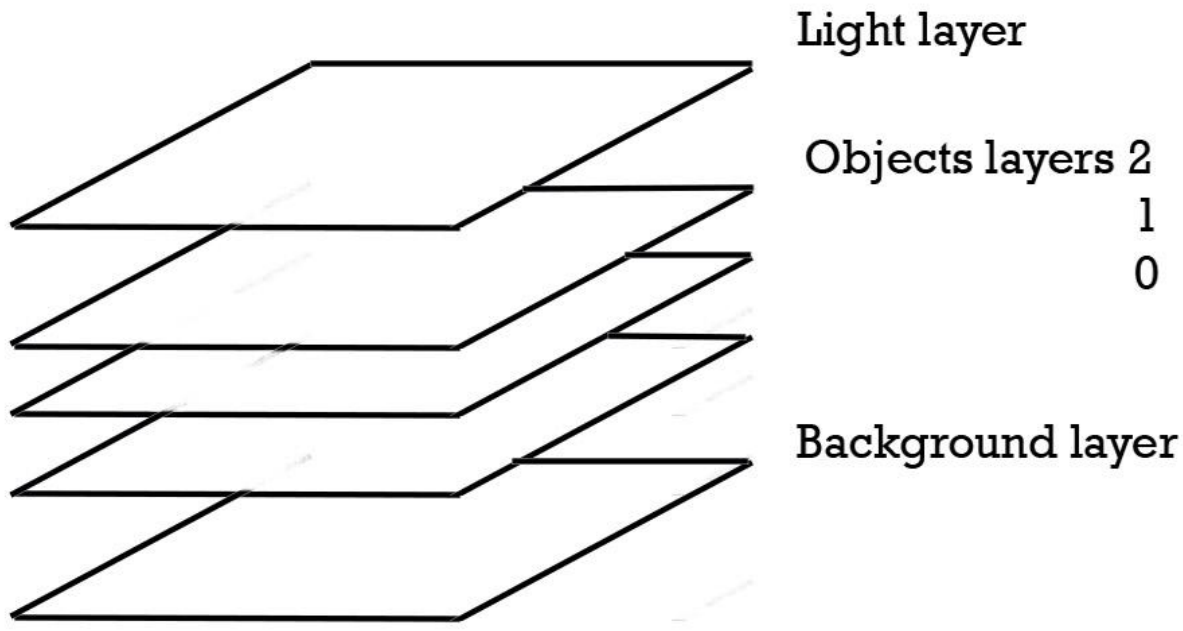
(Sarah Frisken, 2002)

3.4 Rendering

Το rendering[3] είναι η διαδικασία όπου ζωγραφίζουμε μια τελική εικόνα στην οθόνη η οποία μπορεί να έχει συντεθεί από πολλές άλλες εικόνες ή και κώδικα. Είναι μια σχετικά απλή υλοποίηση ειδικά σε μια γλώσσα υψηλού επιπέδου όπως η Java. Περιπλέκεται όμως όταν εισάγουμε έννοιες όπως κάμερα, φωτισμό και layering οι οποίες δεν κατηγοριοποιούνται ακριβώς στο μέρος rendering αλλά είναι αντικείμενα που πρέπει να δουλεύουν μαζί. Και στην περίπτωση αυτής της εργασίας που θέλουμε όλα αυτά να προσαρμόζονται εύκολα σε κάθε περίπτωση χρήσης. Ήταν ένα μέρος της πτυχιακής, που ενώ απαίτησε λίγη υλοποίηση, παρουσίασε τα περισσότερα προβλήματα και δυσκολίες.

Ξεκινώντας από το βασικό rendering, κάθε αντικείμενο τις βιβλιοθήκης (που θα μπορούσε να φανεί στην οθόνη) δηλώνει μια συνάρτηση για το πώς ζωγραφίζετε στην οθόνη και περιέχει μεταβλητές που ορίζουν την θέση του, το βάθος, το ενεργό animation και frame κα. Η σκηνή η οποία είναι μια συλλογή τέτοιων αντικειμένων, φτιάχνει μια τελική εικόνα ζωγραφίζοντας τα όπως ορίζει το καθένα. Αυτή είναι μια πολύ γενική περιγραφή μιας απλής δομής rendering.

Προσθέτοντας layering (επίπεδα) στην σκηνή έχουμε την δυνατότητα να τοποθετούμε αντικείμενα πάνω η κάτω από άλλα δίνοντας την ψευδαίσθηση του βάθους. Αυτό γίνετε ελέγχοντας την σειρά που ζωγραφίζονται τα αντικείμενα. Η σκηνή χωρίζετε σε πέντε επίπεδα. Ένα επίπεδο για την εικόνα του background, τρία επίπεδα για τα αντικείμενα και ένα για τον φωτισμό. Έτσι έχουμε την δυνατότητα όταν προσθέτουμε αντικείμενα στην σκηνή να επιλέγουμε το βάθος που θα βρίσκετε το αντικείμενο πράγμα πολύ χρήσιμο και για τις συγκρούσεις.



Εικόνα 6 Επίπεδα Rendering

[3] http://docs.oracle.com/javase/tutorial/2d/overview/index.html
http://en.wikipedia.org/wiki/2D_computer_graphics

Το αντικείμενο που ζωγραφίζει την τελική εικόνα είναι το Display το οποίο επεκτείνει το αντικείμενο JPanel της Java. Μπορεί να τρέξει σε δικό του νήμα και κατ'επιλογήν να καθαρίζει ή όχι την εικόνα που πρόκειτε να ζωγραφιστεί, άν και όχι απαραίτητο έγινε έτσι για χρησιμοποιηθεί και απο τον GCE στην ζωγραφική.

3.4.1 Camera

Ζωγραφίζοντας όμως μόνο την τελική εικόνα χωρίς να μπορούμε να κινούμαστε ¨μέσα¨ σ'αυτήν είναι χρήσιμο για ελάχιστα παιχνίδια. Αυτήν την δυνατότητα την δίνουν τα αντικείμενα Camera και Cameraman. Το πρώτο λειτουργεί ως δομή η οποία κρατάει μεταβλητές για θέση, διαστάσεις οθόνης και ζούμ. Το δεύτερο αντικείμενο ελένχει μία Camera. Φτιάχνοντας ένα αντικείμενο Cameraman και προσθέτοντας το σε ένα Display έχουμε την δυνατότητα να κινηθούμε, να ζουμάρουμε, να θέσουμε την περιοχή που μπορούμε να κινηθούμε, να θέσουμε την ταχύτητα της κάμερας ακόμη και να ορίσουμε ένα Sprite το οποίο η κάμερα θα ακολουθεί. Επίσης με τα δεδομένα της κάμερας μπορούμε να υπολογίσουμε το οπτικό πεδίο του παίκτη και να εφαρμόσουμε το κατάλληλο culling που αναφέρετε αμέσως πιο κάτω. Για την διαδικασία culling είναι υπεύθυνη η σκηνή στην οποία πρέπει να περνάμε αυτές τις μεταβλητές.

Παράδειγμα χρήσης Display – Cameraman

```
//Φτιάχνουμε ένα αντικείμενο display και το προσθέτουμε σε ενα JFrame. Επίσης αρχικοποιούμε την
//τελική εικόνα στο μέγενθος του frame.

Display display = new Display();
jframe.add(display);
display.createDisplayImage(jframe.getWidth(),jframe.getHeight());
```

```
//Αφού φτιάξουμε ένα αντικείμενο Cameraman πρέπει να του θέσουμε το μέγενθος του viewport.

Cameraman cameraman = new Cameraman();
cameraman.setFovX(display.getWidth());
cameraman.setFovY(display.getHeight());

//άλλες χρήσιμες συναρτήσεις:

cameraman.setMoveArea(minx,minY,maxX,maxY);
cameraman.setMinZoom(zoomVal);
cameraman.setMaxZoom(zoomVal);
```

```
///Για να ακολουθάει μόνη της η κάμερα τον παίκτη καλούμε την παρακάτω συνάρτηση στην
///αρχικοποίηση:

cameraman.target(player);

///και σε κάθε update:

cameraman.follow();
```

```
///Για να χρησιμοποιήσουμε αυτά τα αντικείμενα στο τελικό render:

scene.setViewPort(cameraman.getViewport());
scene.render((Graphics2D) display.getDisplayImage().getGraphics());
```

3.4.2 Potentially visible sets

Ο φωτισμός επίσης ήταν μεγάλο πρόβλημα για το rendering καθώς οι ημιδιάφανες εικόνες κοστίζουν πολύ επεξεργαστικό χρόνο. Τα καρέ ανα δευτερόλεπτο (fps) είχαν μεγάλη διαφορά στην ίδια σκηνή με φωτισμό και χωρίς. Άν και πρόβλημα, ο φωτισμός ήταν το κίνητρο για την πρώτη μεγάλη βελτιστοποίηση της βιβλιοθήκης στο μέρος rendering. Η λύση ήταν να ζωγραφίζετε μόνο το μέρος της σκηνής, τα αντικείμενα και το επίπεδο φωτισμού που φαίνονται στην οθόνη. Αυτή η τεχνική ονομάζετε potentially visible sets [4]και χρησιμοποιείτε πολύ συχνά στα παιχνίδια και δύο και τριών διαστάσεων για την επιτάχυνση του rendering.

Υπάρχουν πολλές παραλλαγές και βελτιστοποιήσεις για αυτόν τον αλγόριθμο που συνήθως χρησιμοποιούνται σε πιο απαιτητικές εφαρμογές οπως για παράδειγμα:

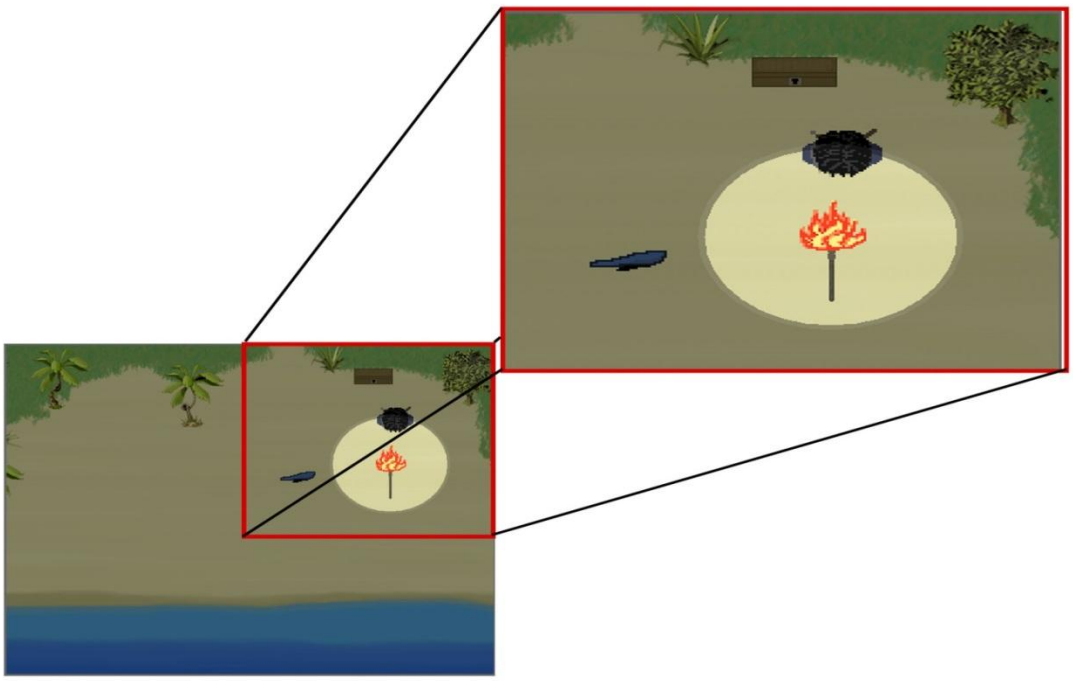
- Viewing frustum culling
  Είναι ο πιό βασικός αλγόριθμος, επιλέγει και ζωγραφίζει μόνο τα αντικείμενα που είναι ορατά στην κάμερα.

- Backface culling
  Αλγόριθμος που χρησιμοποιείτε στο 3D rendering. Αποφασίζει ποιές απο τις επιφάνειες των μοντέλων δέν φαίνονται απο την κάμερα καθώς κοιτάζουν σε αντίθετη κατεύθυνση απο το viewport. Δέν υπάρχει ανάγκη να ζωγραφίζουμε τέτοιες επιφάνειες οπότε μπορούμε έτσι να επιταχύνουμε το rendering.

- Contribution culling
  Αντικείμενα που βρίσκονται πολύ μακριά και δεν συνεισφέρουν σημαντικά στην τελική εικόνα απορρίπτονται.

- Occlusion culling
  Αντικείμενα που κρύβονται ολόκληρα πίσω απο άλλα και συνεπώς δέν είναι ορατά, επίσης δέν ζωγραφίζονται.

[4] http://en.wikipedia.org/wiki/Hidden_surface_determination



Εικόνα 7 Viewing Frustum Culling

3.4.3 Double buffering

Επίσης η βιβλιοθήκη προσφέρει δυνατότητα double buffering που χρησιμοποιείτε για την μείωση του flickering και tearing αποτέλεσμα της "ενδιάμεσης" εικόνας που αντιλαμβάνεται ο χρήστης κατα το rendering. Αυτή η τεχνική χρησιμοποιεί ένα μέρος μνήμης για να αποθηκεύσει τα αποτελέσματα όλων των render συναρτήσεων και τέλος μεταφέρει την τελική εικόνα στην οθόνη.
Μία επιπλέον τεχνική που χρησιμοποιείτε σε συνδυασμό με το double buffering είναι η page flipping. Υπάρχουν δύο buffer οι ποίοι μπορούν και οι δύο να γράψουν στην οθόνη αλλά κάθε φορά ο ένας ζωγραφίζετε και ο άλλος μεταφέρετε στην οθόνη. Όταν η διαδικασία ολοκληρωθεί οι ρόλοι αντιστρέφονται. Αυτή η τεχνική είναι γνωστή και ως ping-pong.

Αντίστοιχα με το double buffering υπάρχει και triple στο οποίο έχουμε τρεις buffer, δύο back-buffer που εναλλάσσονται με ένα front-buffer, και quad buffering που χρησιμοποιείτε σε στερεοσκοπικά render όπου δεσμεύονται δύο buffer για κάθε μάτι.

3.4.4 FPS & Delta Time

Μία λεπτομέρεια που παίζει σημαντικό ρόλο στο ομαλό τρέξιμο του παιχνιδιού και την εξοικονόμηση κύκλων επεξεργαστή , είναι η διατήρηση ενός συνεχούς ρυθμού καρέ ανά δευτερόλεπτο (fps)[5]. Ο ρυθμός που ένα παιχνίδι κάνει update και render πρέπει να είναι ανεξάρτητος της ταχύτητας του επεξεργαστή κάθε συστήματος ώστε να τρέχει πάντα με τον ίδιο ρυθμό. Ο χρόνος που περιμένουμε (sleep) ανάμεσα σε κάθε κύκλο παιχνιδιού πρέπει να διαφέρει για κάθε ταχύτητα

[5] http://en.wikipedia.org/wiki/Frame_rate

συστήματος. Μία έννοια που χρησιμοποιείται σε τέτοιου είδους προβλήματα που συναντώνται όχι μόνο στα γραφικά αλλά και στα δίκτυα και στα hardware είναι η Delta Time. Στην περίπτωση των γραφικών περιγράφει πόσος χρόνος έχει περάσει από το τελευταίο update του παιχνιδιού. Σάν υλοποίηση είναι μία μεταβλητή που κρατάει κάθε φορά τον χρόνο του τελευταίου update ώστε να μπορούμε να υπολογίσουμε πότε θα πρέπει να γίνει το επόμενο. Με αυτήν την τεχνική ο χρόνος ανάμεσα σε κάθε update είναι πάντα ο ίδιος ανεξάρτητα του συστήματος.

3.5 Sprites & Animation

Το animation είναι η εναλλαγή εικόνων που δίνουν την αίσθηση κίνησης. Είναι βασικό συστατικό των ηλεκτρονικών παιχνιδιών και είναι ένα στάδιο πού χρειάζεται πολύ δουλειά. Και απο τον καλλιτέχνη που πρέπει να δώσει την εντύπωση ομαλής κίνησης και απο τον προγραμματιστή καθώς είναι μια επεξεργαστικά ακριβή διεργασία που θέλει προσοχή και συγχρονισμό.

Τα αντικείμενα που αποτελούνται απο μία εικόνα έως πολλά animation και τα οποία μπορούν να μετακινηθούν μέσα σε μια σκηνή και να αλληλεπιδρούν με τον χρήστη και άλλα αντικείμενα ονομάζονται sprites. Η βιβλιοθήκη κατηγοριοποιεί όσα είναι στατικές εικόνες ως Spites και όσα περιέχουν animation με πολλές εικόνες ως Animated Sprite. Χρησιμοποιώντας τον Sprite Editor του GCE μπορούμε εύκολα να δημιουργήσουμε τέτοια αντικείμενα ζωγραφίζοντας καρέ και περιγράφοντας τα animation ως σειρές εικόνων. Το τελικό αρχείο για ένα απλό sprite είναι μια εικόνα ενώ για ένα animated sprite είναι ένα αρχείο το οποίο περιέχει όλα τα καρέ του αντικειμένου και ένα .txt που περιγράφει τα animation. Το αρχείο του animation γράφετε για παράδειγμα έτσι :

```
Idle 0
Walk 0 1 2 3 2 1 4 5 6 5 4 3
Jump 0 7 8 9
```

Όπου το πρώτο μέρος είναι το όνομα του animation και ότι ακολουθεί είναι το κάθε καρέ που το αποτελεί. Έτσι η βιβλιοθήκη μας δίνει την δυνατότητα να φτιάξουμε τέτοια sprite και χωρίς την χρήση του GCE απλά δημιουργώντας ένα αρχείο το οποίο περιέχει όλα τα καρέ και το αρχείο animations.txt όπως το παράδειγμα πιο πάνω.

Έχοντας ένα τέτοιο αντικείμενο έτοιμο μπορούμε να το φορτώσουμε είτε μεμονωμένα είτε ως μέρος μιας σκηνής, η βιβλιοθήκη χρειάζεται μόνο το όνομα του αρχείου και κάνει τα υπόλοιπα μόνη της, φορτώνει όλες τις εικόνες, μεταφράζει και αρχικοποιεί τα animation και καταγράφει το sprite. Οι κλάσεις Sprite και AnimatedSprite δίνουν έτοιμες συναρτήσεις για κίνηση, εναλλαγή animation, rendering, bounds, συμβάντα κα. Μπορούμε εύκολα να προσθέσουμε τέτοια αντικείμενα σε μια σκηνή είτε μέσω του Scene Editor είτε γράφοντας στο αρχείο config.txt:

```
SP:imageName/x/y/w/h/l
```

Για απλά στατικά sprite και,

```
AS:fileName/x/y/l
```

για animated sprites, όπου (x,y) οι συντεταγμένες μέσα στην σκηνή και (l) σε πιό επίπεδο της σκηνής ανοίκει. Τα (imageName) και (fileName) είναι το όνομα του αρχείου εικόνας και το όνομα ενός φακέλου αντίστοιχα απο τα οποία αποτελούνται τα sprite, τα ονόματα χρησιμοποιούνται ως αναγνωριστικά μέσα στο παιχνίδι. Όταν εισάγουμε πολλά ίδια sprite (στην περίπτωση δέντρων) τα οποία έχουν το ίδιο όνομα, η βιβλιοθήκη αναθέτει αυτόματα μοναδικά αναγνωριστικά ώστε να μπορούμε να τα προσπελάσουμε. Έτσι αν φορτώσουμε για παράδειγμα το sprite με όνομα “tree” δεύτερη φορά στην ίδια σκηνή, το τελευταίο αντικείμενο θα πάρει το αναγνωριστικό “tree-2”.

Εικόνα 8: Robin sprite

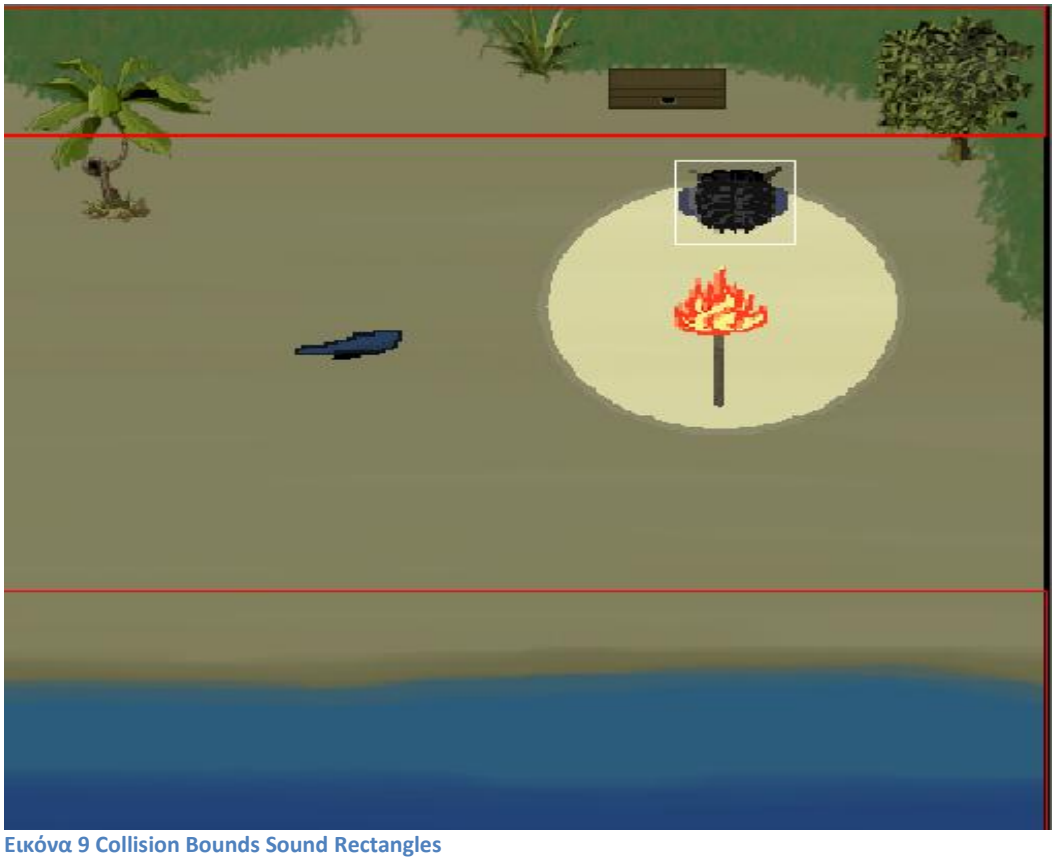
3.6 Ήχος

Ο ήχος[6] είναι από τα σημαντικότερα χαρακτηριστικά που δίνουν ρεαλισμό σε ένα παιχνίδι. Και καθώς τα παιχνίδια χρησιμοποιούν συνήθως πολλούς διαφορετικούς ήχους, απο τους οποίους κάποιοι μπορεί να παίζουν μαζί, κάποιοι να παίζουν με εφέ , άλλοι να παίζουν μία φορά και άλλοι συνεχόμενα, υπάρχει ανάγκη για μεγάλη οργάνωση και συγχρονισμό. Είναι μια διαδικασία χρονοβόρα και στην παραγωγή του ήχου και στον προγραμματισμό, αλλά αν γίνει σωστά μπορεί να δώσει μια ατμόσφαιρα που αλλάζει ριζικά το παιχνίδι.

Η βιβλιοθήκη με τα αντικείμενα SoundEngine και Sound δίνει την δυνατότητα στον χρήστη να φορτώσει ήχους σε ένα pool ή απλά δηλώνοντας τα ονόματα των αρχείων ή ενός φακέλου που περιέχει πολλούς ήχους και προσφέρει δυνατότητες αναπαραγωγής όπως ρύθμιση έντασης, αναπαραγωγή μία φορά ή με επανάληψη και πολλαπλών ήχων. Επίσης ήχοι μπορούν να δηλωθούν και με άλλους δύο διαφορετικούς τρόπους. Ο πρώτος είναι να προσθέσουμε ένα ήχο ώς background στην σκηνή απλά γράφοντας στο config.txt αυτής της σκηνής:

BS:soundname/volume

Όπου “soundname” το όνομα του ήχου και “volume” η ένταση που θέλουμε να παίζει. Ο δεύτερος τρόπος είναι να δηλώσουμε πάλι στο config της σκηνής ένα bound ήχου το όποιο θα παίζει μόνο όταν ο παίχτης ή άλλο αντικείμενο μπεί στα όρια που ορίζει. Αυτή η δυνατότητα είναι χρήσιμη όταν θέλουμε να δώσουμε την ψευδαίσθηση πως πλησιάζουμε σε κάτι. Για παράδειγμα στην εικόνα πιο κάτω η σκηνή περιέχει 2 bound (κόκκινα τετράγωνα) τα οποία εκκινούν ήχους ανάλογα αν ο παίχτης πλησιάσει το δάσος ή την θάλασσα.



Εικόνα 9 Collision Bounds Sound Rectangles

[6] http://docs.oracle.com/javase/tutorial/sound/

Τέτοια bound μπορούν εύκολα να προστεθούν στην σκηνή είτε μέσω του Scene Editor ή απλά δηλώνοντας στο config:

CBSR: soundname/mode/volume#x/y/w/h

Όπου «soundname» το όνομα του αρχείου ήχου, «mode» μπορεί να είναι ή ‘p’ ή ‘l’ για αναπαραγωγή μία φορά η με επανάληψη αντίστοιχα, «volume» η ένταση του ήχου και «x/y/w/h» οι συντεταγμένες και διαστάσεις του bound.

Για το παράδειγμα πιο πάνω:

CBSR:seaB.wav/l/40#0/450/1000/200
CBSR:forestB.wav/l/40#0/0/1000/100

Αυτή η υλοποίηση έγινε χρησιμοποιώντας τα γεγονότα της Java στο αντικείμενο SoundEvent στο οποίο απαντάει ο GameEventListener της βιβλιοθήκης και συνδυάζεται με το αντικείμενο Quadtree της σκηνής που χρησιμοποιεί για την ανίχνευση συγκρούσεων. Αυτή η λειτουργία μπορεί να χρησιμοποιηθεί και ξεχωριστά απο την σκηνή ώστε να μπορεί ο χρήστης να πυροδοτεί γεγονότα σύμφωνα με δικές του συνθήκες. Όπως για παράδειγμα κάποιοι ήχοι μπορεί να συνοδεύονται απο κάποιο animation ή κάποιο διάλογο, ή να παίζει άλλη μουσική κατά την διάρκεια μάχης και άλλη κατα την αλλαγή σκηνής. Επειδή δεν γίνετε να φτιάξουμε συναρτήσεις για κάθε τέτοια περίπτωση, πρέπει αυτά τα αντικείμενα να είναι ευέλικτα και να μην εξαρτώνται απο τα υπόλοιπα κομμάτια της βιβλιοθήκης.

Επειδή αυτα τα bound ήχου όμως μπορούν να ενεργοποιηθούν και απο άλλα sprite εκτός απο αυτό του παίχτη και σε πολλές περιπτώσεις πρέπει μόνο ο παίκτης να επηρεάζει τέτοια γεγονότα, η βιβλιοθήκη παρέχει τα SoundSource. Αυτά τα αντικείμενα δουλεύουν με την ίδια λογική των bound αλλα ενεργοποιούνται μόνο απο τον παίκτη και έχουν μια επιπλέον λειτουργία, να αυξομειώνουν την ένταση του ήχου ανάλογα με το πόσο μακριά είναι ο παίκτης απο την πηγή. Καθώς τα SoundSource είναι πιό κατάλληλα για περιπτώσεις που θέλουμε ζώνες ήχου (δάσος, θάλασσα, άνεμος κλπ.) και πιό καινούργια στην βιβλιοθήκη, τα bound ήχου παρέμειναν για άλλες περιπτώσεις που θέλουμε και τα υπόλοιπα αντικείμενα να συμβάλουν στα γεγονότα της σκηνής. Για παράδειγμα πολλά παιχνίδια χρησιμοποιούν διαφορετικούς ήχους για βήματα ανάλογα με το έδαφος και τέτοιοι ήχοι εκκινούνται απο όλα αντικείμενα της σκηνής που ¨περπατάνε¨ πάνω. Αυτή η διαφοροποίηση μπορεί να γίνει χρησιμοποιώντας τα bound ήχου για να χωρίσουμε την σκηνή ανάλογα με το έδαφος.

Παραδείγμα χρήσης SoundEngine:

```
SoundEngine soundEngine = new SoundEngine();
soundEngine.loadSounds(new File("C:/GameContent/Sounds"));
soundEngine.setVolume("forestB.wav", volume);
soundEngine.playSound("forestB.wav", loop, volume);   //volume: 0 – 100
soundEngine.playSound("steps.wav", play);       //loop: "l" for loop, = "p" for play
```

Παραδείγμα χρήσης SoundEvent (εφόσον έχουμε κάνει implement ως interface τον GameEventListener μας δίνει την συνάρτηση):

```
public void soundEvent(SoundEvent se) {
        soundEngine.playSound(se.getSource(), se.getMode(), se.getVolume());
}
```

Η οποία πυροδοτήτε απο τα SoundEvent

```
SoundEvent se = new SoundEvent(id)  //id: πρέπει να είναι ίδιο με το όνομα του αρχείου ήχου
se.setMode(mode)                    //mode: καθορίζει αν ο ήχος κάνει loop
se.setVolume(volume)                //volume: ένταση
gameEventListener.soundEvent(se);   //εκκίνηση γεγονότος
```

Έτσι με απλές εντολές μπορούμε να φορτώσουμε και να εκκινήσουμε ήχους κάτι που θα χρειαζόταν αρκετά περισσότερο κώδικα καθώς για να να έχουμε δυνατότητες πολλαπλής αναπαραγωγής, επανάληψης και γεγονότα ήχων απαιτείται κάθε ήχος (αντικείμενο Sound) να τρέχει σε ξεχωριστό νήμα και να μπορεί να συνεργαστεί με τα υπόλοιπα αντικείμενα πράγμα που θέλει προσοχή στον συγχρονισμό και στην δέσμευση πόρων.

3.7 Φωτισμός

Σε αντίθεση με τα τρισδιάστατα παιχνίδια όπου ο φωτισμός[7] είναι αναγκαίος σε μία σκηνή για να φανούν τα αντικείμενα τις, τα δισδιάστατα τον χρησιμοποιούν πιο πολύ ως εφέ και καμιά φορά ως λειτουργική έννοια. Για παράδειγμα εναλλαγή μέρας – νύχτας, να περιορίσουμε την ορατότητα του παίχτη ή αν υπάρχουν αντικείμενα του παιχνιδιού που επιδρούν στο φωτισμό της σκηνής (δάδα, φακός κλπ.). Υπάρχουν πολλά χαρακτηριστικά που μπορούν να περιγράψουν ένα φωτισμό όπως χρώμα, ένταση, εμβέλεια, αν είναι σημειακό (Point), κατευθυνόμενο (Directional), η παγκόσμιο (Global).

Το αντικείμενο Light της βιβλιοθήκης περιγράφει τέτοιους φωτισμούς για την σκηνή και μπορεί να χρησιμοποιηθεί άμεσα μόνο με αυτήν, καθώς αυτή παρέχει την υλοποίηση για το πώς θα γίνουν render αυτοί οι φωτισμοί. Για αυτό το σκοπό η σκηνή περιέχει ένα επίπεδο που ζωγραφίζετε τελευταίο στο στάδιο του rendering πάνω απο όλα τα αντικείμενα και το οποίο είναι μια ημιδιάφανη εικόνα που αναπαριστά το σκοτάδι. Κάθε φωτισμός που περιέχει η σκηνή "καθαρίζει" ένα μέρος αυτού του στρώματος ανάλογα με τα χαρακτηριστικά του (θέση, εμβέλεια, τύπος κλπ) για να δώσει την ψευδαίσθηση φωτεινού σημείου και τέλος η εικόνα που θα προκύψει ζωγραφίζετε πάνω στην τελική εικόνα της σκηνής. Μια επιπλέον λεπτομέρεια που υλοποιήθηκε καθώς χρησιμοποιείτε πολύ συχνά στα ηλεκτρονικά παιχνίδια είναι η εναλλαγή μέρας, νύχτας στην σκηνή. Αυτό γίνετε αυξομειώνοντας την διαφάνεια του στρώματος φωτισμού της σκηνής με ρυθμό που θέτει ο χρήστης, όπως και την μέγιστη και την ελάχιστη διαφάνεια για να αποφύγουμε περιπτώσεις όπως εξολοκλήρου σκοτάδι (μέγιστη διαφάνεια) και να πετύχουμε άλλες όπως συννεφιά (ποτέ μηδενική διαφάνεια).
(Σημείωση: Για να κάνει render η σκηνή τους φωτισμούς πρέπει να θέσουμε την μεταβλητή της defaultLighting σε false, αυτό, καθώς και η εναλλαγή μέρας- νύχτας γίνονται είτε μέσω του Scene Editor στις ρυθμίσεις φωτισμού, είτε μέσω του αρχείου config.txt της σκηνής, είτε καλώντας την σχετική μέθοδο του αντικειμένου της σκηνής).

Καθώς οι ημιδιάφανες εικόνες κοστίζουν πολύ στον επεξεργαστή όπως αναφέρθηκε στο κεφάλαιο Rendering, χρησιμοποιήθηκε η τεχνική viewing frustum culling ώστε να ζωγραφίζουμε μόνο ότι φαίνεται στην οθόνη, δηλαδή το επίπεδο φωτισμού να έχει το μέγεθος και να καλύπτει μόνο το ορατό πεδίο του χρήστη και όχι ολόκληρη την σκηνή.

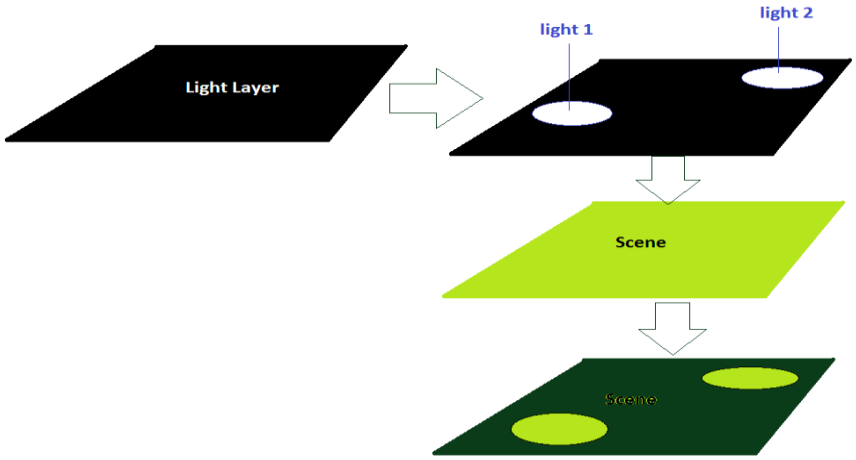
Μπορούμε να δηλώσουμε φωτισμούς στο αρχείο config της σκηνής με την παρακάτω σύνταξη.

LP:id/r/g/b/x/y/range

Αρχίζει πάντα με "L" και ακολουθεί ο τύπος φωτισμού (P: point, D: directional, G: global). Οι τιμές r, g, b ορίζουν το χρώμα του φωτισμού. X, y είναι η θέση του φωτισμού μέσα στην σκηνή και range είναι η εμβέλεια του. Έτσι άν θέλουμε ένα σημειακό κόκκινο φωτισμό με όνομα light1 στην θέση (50,50) με εμβέλεια 100 απλά προσθέτουμε στις δηλώσεις της σκηνής την γραμμή:

LP:light1/255/0/0/50/50/100

[7] http://ncase.me/sight-and-light/
http://www.wholehog-games.com/devblog/2013/06/07/lighting-in-a-2d-game/



Εικόνα 10 Lighting Pipeline

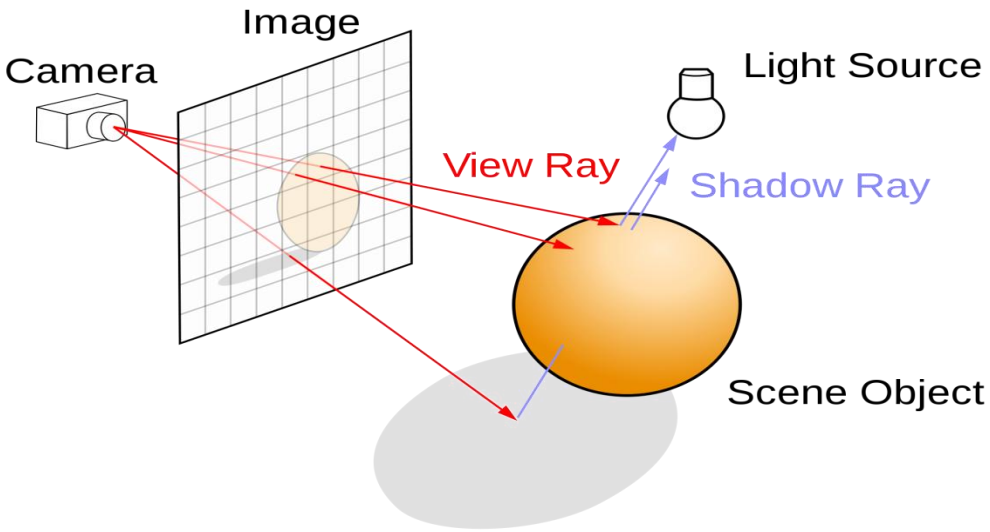
Απο αυτήν την υλοποίηση όμως λείπει κάτι σημαντικό που υπάρχει στα μελλοντικά σχέδια αυτής της εργασίας, οι σκιές. Άν και μπορούμε να δώσουμε την αίσθηση σκιάς μέσα απο τα γραφικά δέν έχουμε την επιλογή για δυναμικές σκιές [8] που αλλάζουν καθώς κινούνται τα αντικείμενα. Η τεχνική με την οποία μπορούμε να το πετύχουμε αυτό λέγετε Ray Casting.

3.7.1 Ray Casting

Η ιδέα του ray casting [9]είναι να ρίχνουμε μία ακτίνα απο ένα σημείο και προς μία κατεύθυνση και να εντοπίζουμε το πρώτο αντικείμενο με το οποίο συγκρούεται. Αυτή η υλοποίηση βρίσκει πολλές εφαρμογές και όχι μόνο στα ηλεκτρονικά παιχνίδια. Ray casting χρησιμοποιούνται στις σκιές, στα βλήματα για τον υπολογισμό του τελικού στόχου, στους φωτισμούς, στην κάμερα για τον υπολογισμό του οπτικού πεδίου και των κρυμμένων επιφανειών κα.

[8] https://www.kickstarter.com/projects/finnmorgan/sprite-lamp-dynamic-lighting-for-2d-art

[9] http://en.wikipedia.org/wiki/Ray_casting



Εικόνα 11 Ray Casting

Ένα απλό παράδειγμα χρήσης είναι οι δυναμικές σκιές, όπου εκκινούμε ακτίνες από την πηγή του φωτός και εντοπίζουμε με ποια σώματα συγκρούονται. Έτσι ξέροντας την απόσταση και την γωνία αυτής της ακτίνας μπορούμε να υπολογίσουμε από ποια σώματα και προς ποια κατεύθυνση θα ζωγραφιστεί σκιά. Με αυτήν την τεχνική μπορούμε να έχουμε σκιές που μεταβάλλονται όσο κινούνται τα αντικείμενα ή οι φωτισμοί.

Ένα άλλο παράδειγμα είναι ο υπολογισμός του τελικού στόχου ενός βλήματος (σφαίρας). Εκκινώντας μία ακτίνα από το όπλο του παίχτη μπορούμε να υπολογίσουμε που θα καταλήξει η σφαίρα πριν ακόμη ο παίχτης πάρει την απόφαση να πυροβολήσει. Αυτό είναι χρήσιμο σε διαδικτυακά παιχνίδια όπου ο συγχρονισμός είναι σημαντικός και η καθυστέρηση εμπλέκεται στην ροή του παιχνιδιού. Πρέπει να σκεφτεί κανείς, πως στον χρόνο που θα κάνει μια σφαίρα να φτάσει στον αντίπαλο, πρέπει κάποιος υπολογιστής πιθανών στην άλλη άκρη του κόσμου να ξέρει πού αυτή η σφαίρα θα καταλήξει. Και αυτό πρέπει να γίνετε για δεκάδες η και εκατοντάδες άτομα που εξυπηρετεί ένα παιχνίδι συνχρόνως.

3.8 Συστήματα Σωματιδίων

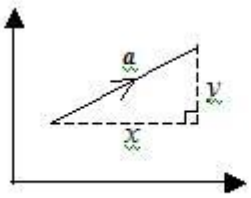
Τα συστήματα σωματιδίων [10]χρησιμοποιούνται για γραφικά αντικείμενα τα οποία δεν έχουν κάποια προκαθορισμένη μορφή ή και σημείο στον χώρο. Για παράδειγμα αέρια, υγρά, χαοτικά συστήματα, εφέ και φυσικά φαινόμενα. Η υλοποίηση τέτοιων συστημάτων είναι λίγο πολύ πάντα η ίδια. Δημιουργούμε σωματίδια (εικονοστοιχεία στο χώρο) με κάποια χαρακτηριστικά (μέγεθος, χρώμα, διαφάνεια) και τα προσθέτουμε σε ένα σύστημα που τα τρέχει με κάποιες ρυθμίσεις (αρχική ταχύτητα, επιτάχυνση, κατεύθυνση, εύρος ζωής κλπ).

Για κάθε ένα από αυτά τα στάδια η βιβλιοθήκη έχει τα αντικείμενα Particle και ParticleSytem αντίστοιχα. Το Particle είναι η δομή που κρατάει την υπόσταση κάθε σωματιδίου που δημιουργούμε. Το ParticleSystem δίνει την δυνατότητα να ορίσουμε την φύση του συστήματος θέτοντας ποίες θα είναι αυτές οι μεταβλητές που θα αρχικοποιεί τα σωματίδια. Με λίγες επιλογές παραμετροποίησης μπορούμε να φτιάξουμε πολλά διαφορετικά συστήματα. Ο Particle System Editor του GCE προσφέρει πολλές επιλογές για να ορίσουμε τα χαρακτηριστικά τέτοιων συστημάτων,

Για την υλοποίηση δημιουργήθηκε ένα ακόμη αντικείμενο, το Vector2F το οποίο αναπαριστά ένα δισδιάστατο διάνυσμα[11] και χρησιμοποιείτε για την θέση, την επιτάχυνση και την ταχύτητα του σωματιδίου καθώς επίσης και για να εφαρμόζουμε δυνάμεις (βαρύτητα, άνεμος κλπ). Αυτό το αντικείμενο περιέχει δύο βασικές συναρτήσεις, τις magnitude και normalize. Η πρώτη υπολογίζει το μέγεθος του διανύσματος χρησιμοποιώντας το πυθαγόρειο θεώρημα και η δεύτερη το ομαλοποιεί στην μονάδα διαιρώντας κάθε συντεταγμένη με με το αποτέλεσμα της πρώτης.

The magnitude of vector **a** is written as $|\mathbf{a}|$.
The magnitude of the vector $\overline{AB}$ is written as $|AB|$.

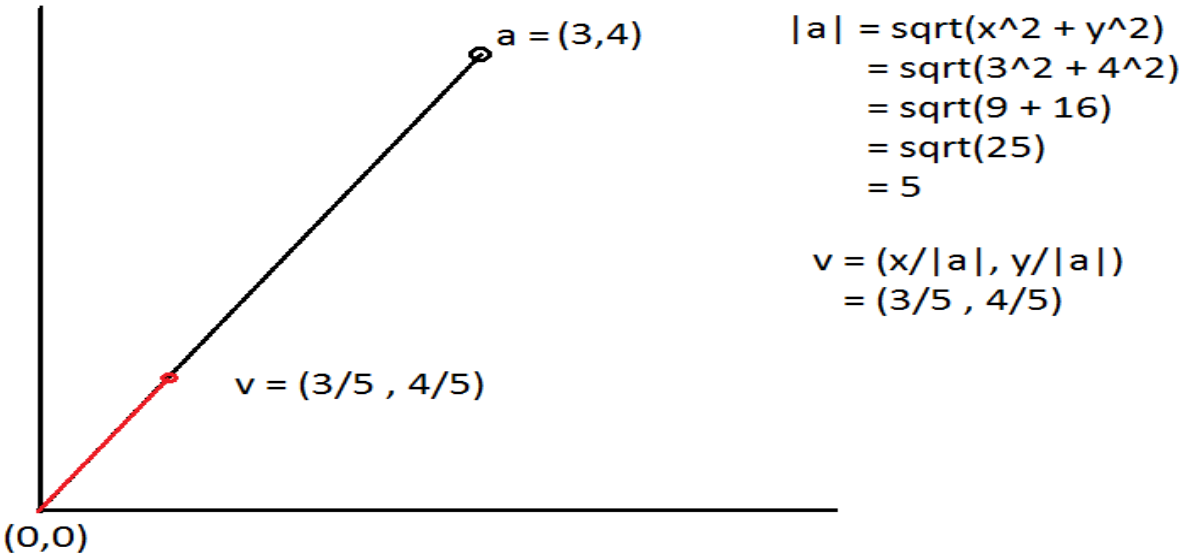
If $\mathbf{a} = \begin{pmatrix} x \\ y \end{pmatrix}$ then the magnitude $|\mathbf{a}| = \sqrt{x^2 + y^2}$ (using the Pythagorean theorem)



Εικόνα 12 Vector magnitude

[10] http://en.wikipedia.org/wiki/Particle_system
http://natureofcode.com/book/chapter-4-particle-systems/

[11] http://www.mathsisfun.com/algebra/vectors.html



Εικόνα 13 Vector normalize

Όταν αρχικοποιούμε ένα σωματίδιο του ορίζουμε την θέση, την κατεύθυνση, αρχική ταχύτητα και επιτάχυνση. Κάνοντας normalize το διάνυσμα της κατεύθυνσης και πολλαπλασιάζοντας το με το διάνυσμα της επιτάχυνσης παίρνουμε ένα νέο διάνυσμα για την επιτάχυνση το οποίο σε κάθε update το προσθέτουμε σε αυτό της ταχύτητας και αυτό με την σειρά του προστίθεται στο διάνυσμα της θέσης. Αν δίναμε ένα διάνυσμα κατεύθυνσης (3,4) σε ένα σωματίδιο όπως το παράδειγμα πιο πάνω και υποθέτουμε ότι το σωματίδιο έχει ταχύτητα 1, αν δεν χρησιμοποιούσαμε την normalize το σωματίδιο σε ένα update θα τηλεμεταφερόταν απο την την θέση (0,0) κατευθείαν στην θέση (3,4).

Μπορούμε να φτιάξουμε τέτοια συστήματα με τον Particle System Editor του GCE ο οποίος τα σώνει σε .txt μορφή και έπειτα να τα προσθέσουμε στην σκηνή ή μέσω του Scene Editor, ή μέσω κώδικα, ή γράφοντας την δήλωση PS:pasticleSystemID στο config.txt της σκηνής. Οπου particlesystemID το όνομα του συστήματος.

Κάτι που λείπει ακόμη από το αντικείμενο Particle System είναι να μπορούμε να ορίσουμε animation για αυτά τα συστήματα, δηλαδή να μεταβάλλονται με τον χρόνο οι τιμές που το καθορίζουν ώστε να μην είναι πάντα το ίδιο. Για παράδειγμα μπορεί να θέλουμε τα σωματίδια να αλλάζουν χρώμα, το σύστημα να κινείτε ή να αλλάζουν οι δυνάμεις που επιδρούν πάνω σε αυτό όσο περνάει ο χρόνος. Όλα αυτά μπορούν να γίνουν μέσω κώδικα καθώς αυτό το αντικείμενο παρέχει συναρτήσεις για να ορίσουμε αυτές τις τιμές, δεν θα ήταν πρακτικό όμως να γίνει έτσι αν επιδιώκουμε κάτι πιο περίπλοκο. Μία λύση είναι να χρησιμοποιήσουμε keyframes όπως αυτά στο 3D animation. Δηλαδή στιγμιότυπα τα οποία περιγράφουν την κατάσταση του συστήματος εκείνη την στιγμή ή την μεταβολή του από την προηγούμενη.

Επόμενο βήμα σε αυτήν την υλοποίηση θα είναι η δυνατότητα δυναμικής αλληλεπίδρασης των σωματιδίων με το περιβάλλον τους. Για παράδειγμα τα σωματίδια να συγκρούονται με εμπόδια της σκηνής ή κάποιες περιοχές της να ασκούν δυνάμεις σε αυτά. Αυτό θα είναι εφικτό με την ολοκλήρωση της μηχανής φυσικής της βιβλιοθήκης αλλα και τα δύο αυτά μέρη ανήκουν ακόμα στην μελλοντική εργασία.

3.9 Η Σκηνή

Η σκηνή είναι η συλλογή όλων των αντικειμένων που συνθέτουν τον κόσμο του παιχνιδιού ο οποίος μπορεί να είναι από ένα δωμάτιο έως και ολόκληρο νησί. Η σκηνή εκτός από δομή για να κρατάει όλα τα αντικείμενα περιέχει και τις συναρτήσεις για να τα κάνει update, να τα ζωγραφίσει, να ελέγξει για συγκρούσεις, να εκκινεί ήχους και να ελέγχει το φωτισμό. Η έννοια σκηνή ή αλλιώς κόσμος (world) ή σύμπαν (universe) σε μερικές περιπτώσεις χρησιμοποιείτε πολύ και απο τους δημιουργούς παιχνιδιών και απο τα περιβάλλοντα και εργαλεία που χρησιμοποιούνται στην διαδικασία. Είναι μία σύνθετη έννοια που ορίζει ένα σύνολο αντικειμένων, γραφικών και μη τα οποία ορίζουν το τελικό αποτέλεσμα.

Η σκηνή είναι υπεύθυνη να κρατάει και να ενημερώνει τα αντικείμενα, να ελέγχει για γεγονότα και να πυροδοτεί τα κατάλληλα event, να ελέγχει τον φωτισμό και τέλος να συνθέτει την τελική εικόνα από όλα τα αντικείμενα ανάλογα την θέαση του χρήστη. Πριν ζωγραφιστεί το κάθε αντικείμενο η σκηνή ελέγχει αν αυτό φαίνεται στην τελική εικόνα βάση της θέσης και το ζούμ του θεατή. Άν το αντικείμενο είναι εκτός πλάνου δεν το ζωγραφίζει καθόλου αλλά το ενημερώνει κανονικά.

Η δομή της σκηνής χωρίζετε σε πέντε επίπεδα.

- Background layer (0): Περιέχει την βασική εικόνα φόντου της σκηνής καθώς και όλα τα bound της.

- Objects layer (1-3): Τρία επίπεδα που περιέχουν όλα τα γραφικά αντικείμενα της σκηνής. Είναι χωρισμένα σε τρείς ομάδες ώστε να ελέγχουμε την σειρά με την οποία ζωγραφίζονται δίνοντας την ψευδαίσθηση βάθους.

- Light layer (4): Είναι το επίπεδο που περιέχει την εικόνα του φωτισμού η οποία υπολογίζετε απο όλα τα αντικείμενα Light της σκηνής και ζωγραφίζετε τελευταία πάνω στην τελική εικόνα.

Η σκηνή για να διαχειριστεί τα γεγονότα ήχου, αλλαγής σκηνής και γενικά συγκρούσεων με bound περιέχει τον ακροατή GameEventListener. Ο χρήστης αν επιθυμεί να διαχειριστεί αυτά τα γεγονότα μπορεί να φτιάξει δικό του αντικείμενο που υλοποιεί τον GameEventListener καθώς αυτό είναι interface και να το ορίσει ως listener στην σκηνή. Η σκηνή μπορεί να τρέξει σε δικό της νήμα, άν και όχι απαραίτητο. Αυτό έγινε όχι μόνο για την καλήτερη κατανομή του φόρτου, αλλα για να μπορούμε να διαφοροποιήσουμε τον αριθμό των ενημερώσεων ανα δευτερόλεπτο απο τον αριθμό των καρέ που ζωγραφίζονται ανα δευτερόλεπτο.

Κάθε τι που περιέχεται στην σκηνή (εκτός ότι προσθέτουμε προγραμματιστικά) περιγράφετε στον αρχείο config.txt της. Αυτό το αρχείο περιέχει τις δηλώσεις για το πώς θα δομηθεί η σκηνή που θα τοποθετηθούν τα αντικείμενα, με ποιους ρόλους θα συνδεθούν, τις ρυθμίσεις των φωτισμών κα. Ένα παράδειγμα τέτοιο αρχείου είναι το παρακάτω το οποίο περιέχει δήλωση για κάθε πιθανό τύπο αντικειμένου, όλες αυτές τις δηλώσεις είναι αυτές που περιγράφτηκαν στα προηγούμενα κεφάλαια.

Αρχείο config.txt σκηνής

```
W:3000                  //μέγενθος της σκηνής
H:3000
defaultLight:f          //ρυθμίσεις φωτισμού («f» για να ενεργοποιήσουμε τους φωτισμούς)
shadeOpacity:100        //αρχική ορατότητα
shadeColor:0,0,0        //χρώμα σκοταδιού
shadePerTime:3.0        // χρόνος εναλλαγής μέρας - νύχτας
```

```
//NO COMMENTS ABOVE THIS LINE//        // Σχόλια με «//»

//Sprites(id/x/y/layer):               // Δηλώσεις στατικών sprite

SP:tree/0/0/0
SP:treev2/124/7/0
SP:treev3/907/0/0
SP:palm/110/223/0
SP:palm/250/250/0
SP:palm2v/225/9/0
SP:palm3v/443/37/0
SP:grass/151/161/0
SP:grass2v/56/218/0
SP:grass3v/693/-6/0
SP:chest/764/47/1
SP:asterias/94/357/2

//Animated Sprites(id/x/y/layer):     // Δηλώσεις animated sprite

AS:player/788/120/0
AS:butterfly/638/231/0
AS:fire/792/191/0

//Particle Systems(id):               // Δηλώσεις συστημάτων σωματιδίων

PS:waterfall

//Collision bounds(type:id/mode/volume#x/y/w/h):    //Δηλώσεις collision bounds

CBSR:seaB.wav/l/30#1/300/1000/400
CBSR:forestB.wav/l/35#0/0/1000/250
CBER:scene2/380.0/670.0#603/0/100/50

//Lights(id/r/g/b/x/y/range):         // Δηλώσεις φωτισμών

LP:l1/255/255/255/822.0/236.0/200

//Roles(AnimatedSpriteId:RoleName)    // Αντιστίχηση ρόλων σε sprite
RL:butterfly:butterflyRole

//Player(spriteId)
PL:player                             // Δήλωση sprite του παίκτη
```

3.9.1 Procedural Generation

Άν και η βιβλιοθήκη δεν είναι για τρισδιάστατα παιχνίδια, πιστεύω πως πρέπει να γίνει αναφορά σε μία σημαντική τεχνική που χρησιμοποιείτε πολύ τελευταία στην δημιουργία τρισδιάστατων σκηνών και όχι μόνο, η οποία λέγετε Procedural Generation[12]. Κάποιοι απο τους πιό γνωστούς τίτλους ηλεκτρονικών παιχνιδιών τοποθετούν τον παίκτη σε κόσμους που απλώνονται απο 30.000 έως 60.000 τετραγωνικά μίλια. Πράγμα εκπληκτικό άν σκεφτεί κανείς πόση δουλειά χρειάζεται για να 'γεμίσουν' αυτοί οι κόσμοι με φυσικά τοπία, πόλεις, ανθρώπους, αντικείμενα και όλες τις λεπτομέρειες που θα κάνουν αυτόν τον κόσμο πειστικό. Η τεχνική του Procedural Generation χρησιμοποιεί αλγόριθμους για να παράγει τέτοιο περιεχόμενο. Μπορούμε να περιγράψουμε στον υπολογιστή κάποιους κανόνες με τους οποίους θα δομεί αντικείμενα και κάποια χαρακτηριστικά που θα έχει την ελευθερία να παραμετροποιεί. Έτσι μπορούμε να παράγουμε αλγοριθμικά τέτοια αντικείμενα τα οποία μοιάζουν αλλά είναι όλα διαφορετικά. Για παράδειγμα μπορούμε να φτιάξουμε ένα αλγόριθμο που περιγράφει την μορφή ενός δέντρου έχοντας ώς μεταβλητές το ύψος, πάχος και μήκοι των κλαδιών, την πυκνότητα και κατανομή των φύλλων κα. Δίνοντας τυχαίες τιμές σε αυτές τις μεταβλητές μπορούμε κάθε φορά να παράγουμε ένα διαφορετικό δέντρο με αποτέλεσμα να μπορούμε να δημιουργήσουμε μεγάλα δάση χωρίς να μοντελοποιήσουμε χειροκίνητα τέτοια αντικείμενα. Αυτή η τεχνική χρησιμοποιείτε και για την δημιουργία εδάφους, πόλεων, αντικειμένων, ανθρώπων ακόμη και τον συνδυασμό όλων αυτών. Αυτοματοποιεί σε μεγάλο βαθμό ίσως την πιο χρονοβόρα διαδικασία στην δημιουργία παιχνιδιών αλλά επίσης δίνει την δυνατότητα στον παίκτη να έχει διαφορετική εμπειρία κάθε φορά που παίζει το ίδιο παιχνίδι. Καθώς το περιεχόμενο δημιουργείται αλγοριθμικά με τυχαίες τιμές κάθε φορά ο κόσμος θα είναι διαφορετικός

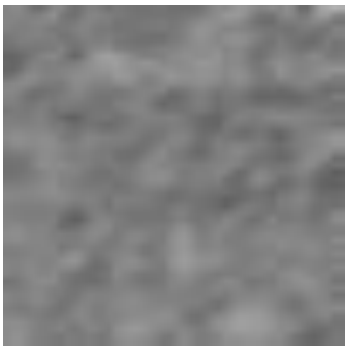
3.9.2 Height Maps & Diamond Square

Η βιβλιοθήκη περιέχει μια κλάση, την HeightMap[13] η οποία είναι ένα πολύ απλό παράδειγμα τέτοιων αλγορίθμων και χρησιμοποιείται για την δημιουργία τυχαίου εδάφους[14]. Ένα heightmap είναι μία ασπρόμαυρη εικόνα όπου κάθε pixel αναπαριστά το ύψος του εδάφους σε κάθε το σημείο. Για να φτιάξουμε ένα αντικείμενο HeightMap πρέπει να προσδιορίσουμε το πλάτος, το ύψος, το ελάχιστο και το μέγιστο επίπεδο. Ο αριθμός των επιπέδων ορίζει την ανάλυση του βάθους ενώ η ανάλυση του heightmap καθορίζει πόσα pixel του εδάφους αναπαριστούν 1 pixel του heightmap.

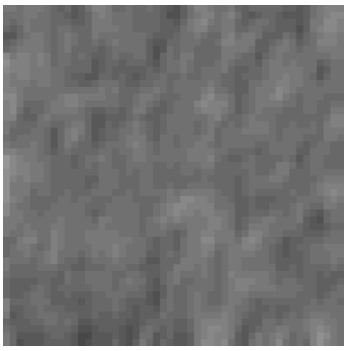
[12] http://en.wikipedia.org/wiki/Procedural_generation
http://pcgbook.com/

[13] http://en.wikipedia.org/wiki/Heightmap

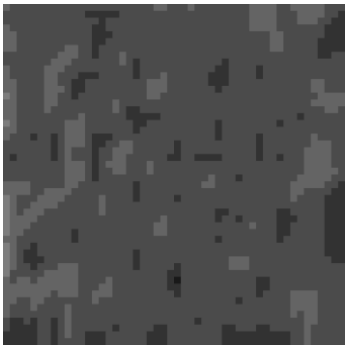
[14] http://www.gameprogrammer.com/fractal.html



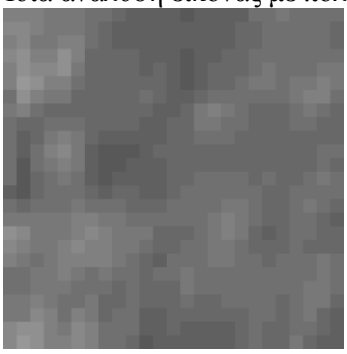
Heightmap με μεγάλη ανάλυση εικόνας και βάθους.



Ίδια ανάλυση με την πρώτη αλλά μικρότερο βάθος.

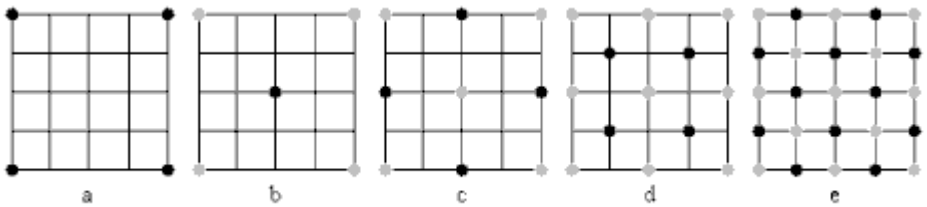


Ίδια ανάλυση εικόνας με πολύ μικρότερο βάθος.



Ίδιο βάθος πολύ μικρότερη ανάλυση.

Το αντικείμενο HeightMap δημιουργεί αυτές τις εικόνες πρώτα γεμίζοντας ένα δισδιάστατο πίνακα με τυχαίους αριθμούς από το ελάχιστο μέχρι το μέγιστο επίπεδο. Έπειτα εφαρμόζει μια εξομάλυνση (smoothing) , τον αλγόριθμο diamond square και τέλος ένα δεύτερο smoothing. Η συνάρτηση για την εξομάλυνση υπολογίζει για κάθε pixel μία μέση τιμή απο τα γειτονικά και του θέτει αυτήν την τιμή. Έτσι δίνουμε τη αίσθηση συνέχειας του εδάφους. Ο αλγόριθμος diamond square [15]επιλέγει κάθε φορά τα άκρα του χάρτη υπολογίζει την μέση τιμή και την θέτει στο κέντρο αυτών των άκρων. Έπειτα διαιρεί τον χάρτη σε τέσσερα μέρη και επαναλαμβάνει την ίδια διαδικασία όπως φαίνεται στην εικόνα πιο κάτω.



Εικόνα 14 Diamond Square Algorithm

[15] http://en.wikipedia.org/wiki/Diamond-square_algorithm

Τέτοια height maps χρησιμοποιούνται και στα δισδιάστατα και στα τρισδιάστατα παιχνίδια και έχουν και άλλες χρήσεις εκτός την δημιουργία εδάφους. Μπορούν να χρησιμοποιηθούν για bump mapping και displacement mapping τα οποία αυξάνουν σε μεγάλο βαθμό την λεπτομέρεια σε 3D μοντέλα. Η δημιουργία τέτοιων maps δέν είναι υποχρεωτικά τυχαία, υπάρχουν πολλά προγράμματα στα οποία αυτά μπορούν να ζωγραφιστούν με το χέρι.

Παράδειγμα χρήσης HeightMap:

```
//Αρχικοποιούμε το αντικείμενο με πλάτος, μήκος, ελάχιστο και μέγιστο επίπεδο:
HeightMap heightMap =  new HeightMap(int width, int height, int minLevel, int maxLevel);

heightMap.generateRandom();              // Δημιουργούμε ένα τυχαίο heightmap.

heightMap.draw(Graphics2D g2d);          // Μπορούμε να το ζωγραφίσουμε δίνοντας ένα
                                         // αντικείμενο graphics.

heightMap.save(String path);             // Μπορούμε να σώσουμε το heightmap σε ένα directory.
```

3.10 Έλενχος Ουδέτερων Αντικειμένων

Ένας από τους τρόπους να δώσουμε ρεαλισμό σε μια σκηνή, είναι να δώσουμε ζωή στα αντικείμενα που περιέχει, να δώσουμε την ψευδαίσθηση στον παίκτη ότι δεν είναι μόνος. Μπορούμε να το πετύχουμε αυτό αυτοματοποιώντας κινήσεις, animation και ήχους για αντικείμενα που δεν ελέγχονται από τον χρήστη. Η βιβλιοθήκη έχει αυτήν την δυνατότητα διαβάζοντας ρόλους από αρχεία κειμένου που μπορεί εύκολα να συντάξει ο χρήστης και ελέχνοντας αντικείμενα όπως animated sprites.

Το αντικείμενο SpriteController μπορεί να πάρει τον έλεγχο sprite και να εφαρμόζει τις εντολές των ρόλων που τα συσχετίσαμε. Δέν χρησιμοποιείτε ως εργαλείο για τεχνητή νοημοσύνη αλλά για περιπτώσεις που θέλουμε προκαθορισμένες κινήσεις και συμβάντα που δεν επηρεάζονται από τον παίκτη. Για να συντάξουμε ένα αρχείο ρόλου έχουμε πέντε εντολές, ότι βρίσκετε μετά την πρώτη άνω-κάτω τελεία είναι ορίσματα του χρήστη.

- setSpeed:speed : Θέτει την ταχύτητα που κινείται το αντικείμενο.
- moveTo:x:y : Ορίζει μια θέση που θα κινηθεί το αντικείμενο. Η κίνηση γίνετε σταδιακά με ταχύτητα που θέτουμε όπως πρήν.
- chAnim:animationId : Αλλάζει το ενεργό animation του αντικειμένου.
- playSound:soundId : Εκκινεί έναν ήχο.
- wait:time : Περιμένει για κάποιο χρόνο.

Ένα τέτοιο αρχείο ρόλου μπορούμε να το συνδέσουμε με κάποιο sprite είτε προγραμματιστικά μέσω της βιβλιοθήκης, ή δηλώνοντας στο config.txt της σκηνής: RL:spriteID:roleID όπου spriteID είναι το όνομα του sprite και roleID το όνομα του ρόλου. Άν επιλέξουμε τον δεύτερο τρόπο δέν χρειάζετε απο εμάς να φτιάξουμε SpriteController ούτε να ορίσουμε ρόλους προγραμματιστικά, ασχολείτε η βιβλιοθίκη με αυτό.

Μία πιθανή μελλοντική υλοποίηση για αυτό το αντικείμενο είναι η δυνατότητα να του περνάμε δεδομένα που δεχόμαστε απο άλλους παίχτες μέσω δικτύου ώστε να ελέγχουμε τα sprite που αναπαριστούν (avatar). Δηλαδή να εξομοιώνει την είσοδο των υπόλοιπων παιχτών για αντικείμενα που δέν έχουμε εμείς τον έλεγχο. Με την ίδια λογική μπορεί να χρησιμοποιηθεί και ως το μέσο με το οποίο μια μελλοντική υλοποίηση τεχνητής νοημοσύνης θα μπορεί να παίρνει τον έλεγχο αντικειμένων μεταφράζοντας αποφάσεις σε εντολές όπως πιο πάνω.

Παραδείγμα χρήσης SpriteController:

```
Role role = contentLoader.loadRole(roleId);
SpriteController spriteCntrl = new SpriteController();

sprite.setRole(role);
spriteCntrl.addSprite(sprite);
spriteCntrl.start();
```

Παραδείγμα σύνταξης αρχείου Role:

```
setSpeed:5
chAnim:walk
moveTo:200:100
chAnim:idle
```

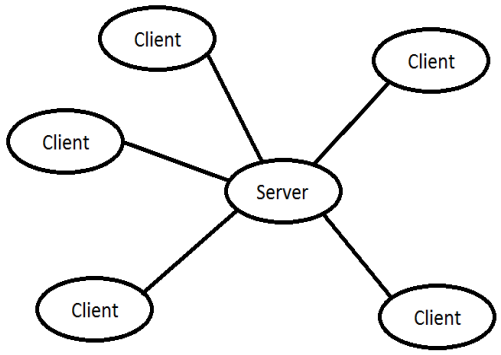
```
wait:2000
chAnim:walk
moveTo:50:50
chAnim:idle
wait:3000
```

3.11 Δικτύωση

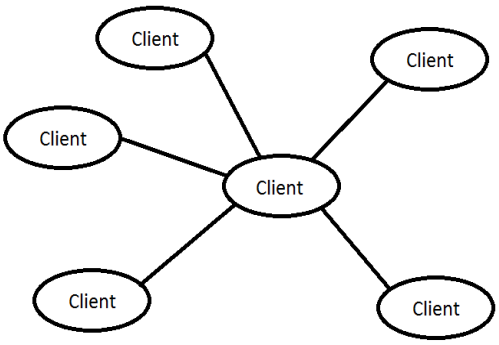
Η δικτύωση δεν είναι καινούργια έννοια για τα ηλεκτρονικά παιχνίδια αλλά μόνο από πρόσφατα χρησιμοποιείτε σε τόσο μεγάλο βαθμό και από άποψη έκτασης και από πολυπλοκότητας. Πλέον υπάρχουν παιχνίδια με εξαψήφιο αριθμό κοινού, που όλοι μπορούν να αλληλεπιδράσουν σε επίπεδα μεγαλύτερα από απλή συνομιλία. Πλέον υπάρχουν παιχνίδια που υποστηρίζουν την δική τους οικονομία, τους δικούς τους νόμους ιδιοκτησίας ψηφιακών αντικειμένων ακόμη και παιχνίδια που μπορούν να παράγουν πραγματικό εισόδημα. Τέτοια παιχνίδια εξαρτώνται σχεδόν αποκλειστικά από την δικτύωση.

Υπάρχουν πολλές διαφορετικές μέθοδοι υλοποίησης για δικτύωση στα ηλεκτρονικά παιχνίδια ανάλογα με τις απαιτήσεις. Υπάρχουν πραγματικού χρόνου και μη, από δύο παίχτες μέχρι χιλιάδες, υλοποιήσεις με ενδιάμεσο εξυπηρετητή και όχι (peer-to-peer) κλπ. Μία γνωστή μέθοδος και που χρησιμοποιεί η βιβλιοθήκη είναι τα εικονικά δωμάτια που επιτρέπουν την σύνδεση από δύο έως μερικών δεκάδων παικτών. Είναι μια υλοποίηση που δεν απαιτεί έναν ενδιάμεσο και σταθερό εξυπηρετητή καθώς το δωμάτιο δημιουργείται και διαχειρίζεται από έναν από τους παίκτες με το μειονέκτημα ότι πρέπει να γίνει γνωστή στους υπόλοιπους η διεύθυνση του διαχειριστή. Αν και μη απαραίτητος ένας εξυπηρετητής θα έλυνε αυτό το πρόβλημα καθώς θα βοηθούσε στο ‘ταίριαγμα’ παιχτών.

Το μέρος της δικτύωσης σε αυτήν την εργασία είναι ακόμη σε πολύ αρχικό στάδιο αλλά παρέχει βασικές υλοποιήσεις σύνδεσης και διαχείρισης εικονικών δωματίων καθώς επίσης και ακροατή συμβάντων για γεγονότα δικτύου. Αυτά τα αντικείμενα χρειάζονται ακόμη αρκετό κώδικα από την μεριά του χρήστη για κάποιες λεπτομέρειες συχνά αναγκαίες στα ηλεκτρονικά παιχνίδια. Όπως για παράδειγμα διανομή τυχαίων αριθμών για τυχαία γεγονότα μέσα στο παιχνίδι που όμως πρέπει να συμφωνούν με όλους τους παίχτες, συγχρονισμός γεγονότων ανάλογα την καθυστέρηση, αναπαράσταση δεδομένων και η δυνατότητα δικτύωσης επικεντρωμένη σε ένα εξυπηρετητή. Υπάρχει η δυνατότητα να αυτοματοποιήσουμε μεγάλο μέρος αυτών των διεργασιών ώστε να κάνουμε ακόμη πιο απλή την υλοποίηση για τον χρήστη.



Εικόνα 15 Server Centered Topology



Εικόνα 16 Room Topology

Πιο πάνω είναι οι τοπολογίες δικτύου για πελάτες – εξυπηρετητής και εικονικού δωματίου. Η μόνη τους διαφορά είναι ότι στο εικονικό δωμάτιο ένας από τους χρήστες αναλαμβάνει τον ρόλο του εξυπηρετητή ή τυχαία ή μετά από συμφωνία. Αν και μοιάζουν χρησιμοποιούνται για διαφορετικές υλοποιήσεις. Σταθερός εξυπηρετητής χρησιμοποιείται για παιχνίδια με πολλούς παίχτες οι οποίοι διατηρούν μια πρόοδο μέσα στο παιχνίδι και συνεπώς δεδομένα πρέπει να διατηρούνται και χωρίς ο χρήστης να είναι συνδεδεμένος. Δωμάτια είναι πιο πολύ για παιχνίδια που έχουν την μορφή ματς. Δηλαδή ο χρήστης υπάρχει μόνο όσο διαρκεί το παιχνίδι.

(Harold)

3.12 Είσοδος στο 3D

Η δημιουργία τρισδιάστατων γραφικών χωρίζεται σε τρία στάδια:

Modeling: Η δημιουργία αρχείων τρισδιάστατων μοντέλων απο την μορφή αντικειμένων. Αυτό γίνετε ή με την χρήση κάποιου εργαλείου 3D modeling, ή με το σκανάρισμα κάποιου αντικειμένου, με την τεχνική του procedural generation ή ακόμη και με φυσική προσομοίωση.

Layout & Animation: Η διαδικασία στην οποία υπολογίζουμε πώς πρέπει να τοποθετηθούν τα μοντέλα μέσα στην σκηνή ανάλογα την θέση, το μέγεθος, το animation κλπ.

Rendering: Η διαδικασία στην οποία παράγουμε την τελική εικόνα απο τα αντικείμενα, τους φωτισμούς, το είδος των επιφανειών και άλλα δεδομένα.

Αν και η βιβλιοθήκη δεν μπορεί να χρησιμοποιηθεί ακόμα για τρισδιάστατα παιχνίδια υπάρχουν μερικά αντικείμενα τα οποία φτιάχτηκαν για να χρησιμοποιηθούν σε μελλοντική εργασία. Κάθε τρισδιάστατο μοντέλο αποτελείτε από σημεία στον χώρο (vertices), επιφάνειες (faces) και κατευθύνσεις επιφανειών (normals). Υπάρχουν πολλά ακόμη χαρακτηριστικά όπως textures, materials, normal maps, shader maps, diffuse maps τα οποία καθορίζουν πώς χρωματίζονται και πώς αλληλεπιδρούν με φωτισμούς αυτά τα μοντέλα. Επίσης υπάρχουν πολλές μορφές αρχείων που περιγράφουν μοντέλα όπως obj, 3ds, fbx, x3d, ply κλπ.

Τα αντικείμενα που χρησιμοποιούνται απο την βιβλιοθήκη για να δομήσουν ένα μοντέλο είναι:

- Vector3f: Τρείς float συντεταγμένες που χρησιμεύουν στην αναπαράσταση και των vertices και των normals.
- Face: Κάθε επιφάνεια (τρίγωνο) αποτελείτε απο τρία vertices και τρία normals.
- Model: Η δομή στην οποία αποθηκεύουμε κάθε πληροφορία που χρειαζόμαστε για να αναπαραστήσουμε το μοντέλο.
- ObjLoader: Αντικείμενο το οποίο φορτώνει τρισδιάστατα μοντέλα απο .obj αρχεία.

Ένα obj αρχείο έχει την παρακάτω δομή:

```
v     1.012786    -2.323534    1.045964    1.0
v     0.354364    0.435672     -1.549623   1.0
v     0.567345    0.235623     -0.146385   1.0
.
.
.

vn    0.285634    0.982385     -0.239813
vn    0.543965    -0.138742    0.459634
vn    0.238512    0.972356     -0.814095
.
```

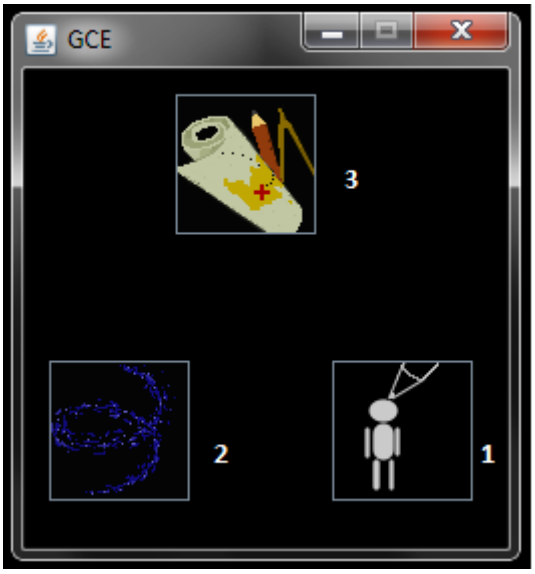
.
.
f 23555//23555 23578//23578 23556//23556
f 23576//23576 22567//22567 21876//21876
f 21372//21373 21345//21345 20365//20365
.
.
.

Κάθε σειρά που αρχίζει με “v” αναπαριστά ένα vertice με κάθε αριθμό που ακολουθεί να ορίζει τις συντεταγμένες x,y,z ενώ ο τέταρτος αριθμός είναι προαιρετικός με default 1.0 και ορίζει το χρώμα του vertice. Οι σειρές με “vn” αναπαριστούν ένα vertex normal με τους αριθμούς να ορίζουν την κατεύθυνση του διανύσματος και οι σειρές με “f” ορίζουν μιά επιφάνεια. Οι αριθμοί που ακολουθούν το “f” είναι οι θέσεις των vertices και των normals, vertexIndexX//normalIndexX, vertexIndexY//normalIndexY, vertexIndexZ//normalIndexZ. Έτσι με τρεις συντεταγμένες ορίζουμε ένα τρίγωνο και με πολλά τρίγωνα μπορούμε να περιγράψουμε έως και πολύπλοκα τρισδιάστατα μοντέλα.

Τα τρισδιάστατα γραφικά είναι ένα περίπλοκο θέμα και ανοίκει στα μακρινά σχέδια αυτής της εργασίας καθώς είναι ατομική και δεν χρησιμοποιείται καμία επιπλέον βιβλιοθήκη. Έγινε όμως ένα πρώτο βήμα από περιέργεια, πειραματισμό και εξοικείωση.

3.13 Οδηγός χρήσης GCE

Με την εκκίνηση του προγράμματος εμφανίζετε το παρακάτω interface απο το οποίο μπορούμε να τρέξουμε τους sub-editor.



Εικόνα 17 Διεπαφή GCE

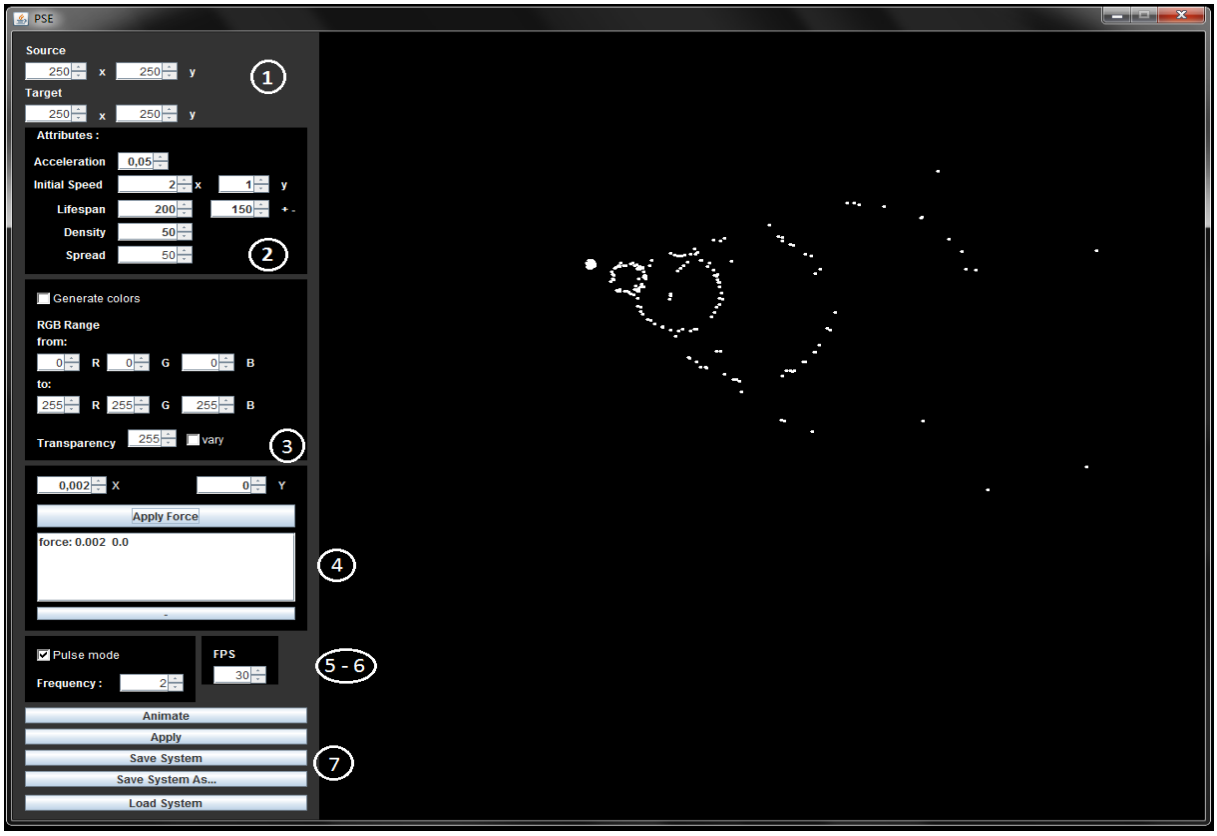
1. Sprite Editor: εργαλείο σχεδίου και animation.

2. Particle Editor: εργαλείο για την δημιουργία συστημάτων σωματιδίων.

3. Scene Editor: εργαλείο για την κατασκευή σκηνών.

Όταν τρέξουμε το πρόγραμμα για πρώτη φορά θα μας εμφανίσει το παρακάτω παράθυρο. Δίνουμε την διαδρομή που θέλουμε να αποθηκεύουμε όλη την δουλειά μας. Εκεί θα δημιουργηθεί το δέντρο αρχείων απο το κεφάλαιο 3.1.1 .

Particle System Editor



Εικόνα 18 Διεπαφή Particle System Editor

Πάνελ 1 : Τα πεδία source (x, y) και target (x, y) ορίζουν τα σημεία απο όπου δημιουργούνται και που κατευθύνονται τα σωματίδια αντίστοιχα. Αν αυτά τα σημεία συμπίπτουν τότε τα σωματίδια εκτοξεύονται ομοιόμορφα προς όλες τις κατευθύνσεις.

Πάνελ 2 : Περιέχει κυρίως τα χαρακτηριστικά των σωματιδίων.

- Acceleration : Η επιτάχυνση.
- Initial speed x, y : Η αρχική ταχύτητα για τον κάθε άξονα.
- Lifespan : Ο χρόνος ζωής των σωματιδίων. Το πεδίο (+-) στα αριστερά του ορίζει πόσο θα μεταβάλετε αυτός ο χρόνος για το κάθε ένα με τυχαίες τιμές.
- Density : Το πλήθος των σωματιδίων που γεννιούνται σε κάθε κύκλο.
- Spread : Το πόσο θα αποκλίνουν τα σωματίδια απο την προκαθορισμένη κατεύθυνση.

Πάνελ 3 : Περιέχει ρυθμίσεις για τα χρώματα. Έχοντας ενεργοποιημένο το checkbox Generate colors το σύστημα θα απονέμει τυχαία χρώματα για τα σωματίδια μέσα στα όρια RBG που ορίζουμε στα παρακάτω textfield, αλλιώς το επιλεγμένο χρώμα θα είναι το τέλος αυτού του ορίου. Έτσι τα πεδία κάτω απο το (from:) ορίζουν την αρχή αυτού του ορίου και τα πεδία κάτω απο το (to:) ορίζουν το τέλος του. Το πεδίο transparency μας αφήνει να ορίσουμε την διαφάνεια των σωματιδίων ενω το πεδίο vary αν είναι ενεργό επιλέγει τυχαία το βαθμό διαφάνειας μέχρι την τιμή του transparency.

Πάνελ 4 : Χρησιμοποιείτε για να προσθέτουμε δυνάμεις στο σύστημα. Τα πεδία x και y ορίζουν την ένταση της δύναμης σε κάθε άξονα. Με το κουμπί Apply Force προσθέτουμε την δύναμη στο σύστημα και κάτω απο την λίστα το κουμπί (-) αφαιρεί την επιλεγμένη απο αυτήν.

Πάνελ 5 : Με το checkbox Pulse Mode ενεργοποιημένο το σύστημα θα παράγει μόνο ανά περιόδους με συχνότητα που ορίζουμε στην τιμή του πεδίου Frequency.

Πάνελ 6 : Θέτουμε τα καρέ ανά δευτερόλεπτο.

Πάνελ 7 :

- Animate: Τρέχουμε το σύστημα με τις ορισμένες ρυθμίσεις.
- Apply: Εφαρμόζει τις αλλαγές στο σύστημα.
- Save System: Σώνει το σύστημα.
- Save System As..: Σώνει το σύστημα με διαφορετικό όνομα.
- Load System: Φορτώνει ένα σύστημα απο ένα αρχείο text.

Sprite Editor

Εικόνα 19 Διεπαφή Sprite Editor

1: Επιλογές για νέο αρχείο, σώσιμο και φόρτωση υπάρχον. Για να φορτώσουμε Sprite επιλέγουμε την εικόνα, για AnimatedSprite επιλέγουμε ολόκληρο τον φάκελο. Η μετατροπή απο απλό Sprite σε Animated μπορεί να γίνει αργότερα και αυτόματα απο τον Editor άν προσθέσουμε animation.

2: Εργαλεία ζωγραφικής με επιλογές για χρώμα, διαφάνεια και μέγενθος. Μπορούμε να ζωγραφίσουμε και κατα την διάρκεια που γίνετε animate το Sprite.

3: Με το Set Guideline Image μπορούμε να ορίσουμε την ενεργή εικόνα ώς οδηγό για να ζωγραφίσουμε νέο καρέ. Η εικόνα θα ζωγραφίζετε μόνιμα με διαφάνεια πάνω στην οποία μπορούμε να ζωγραφίσουμε νέα πόζα. Με το Clear Guideline Image καθαρίζουμε την εικόνα οδηγό.

4: Λίστα των animation που περιέζει το Sprite. Με τα κουμπια + και – μπορούμε να προσθέσουμε και να διαγράψουμε το επιλεγμένο.

5: Λίστα των frame που περιέχει το επιλεγμένο animation. Με το + προσθέτουμε το επιλεγμένο καρέ απο το scroll-bar του image set πιο κάτω. Με το - αφαιρούμε το frame απο το animation, όχι απο το Image set του Sprite.

6: Με το κουμπί Animate τρέχουμε το επιλεγμένο animation.

7: Το image set είναι το σύνολο των εικόνων που περιέχει το Sprite. Μπορούμε να προσθέσουμε και να αφαιρέσουμε με το + και - .

Scene Editor

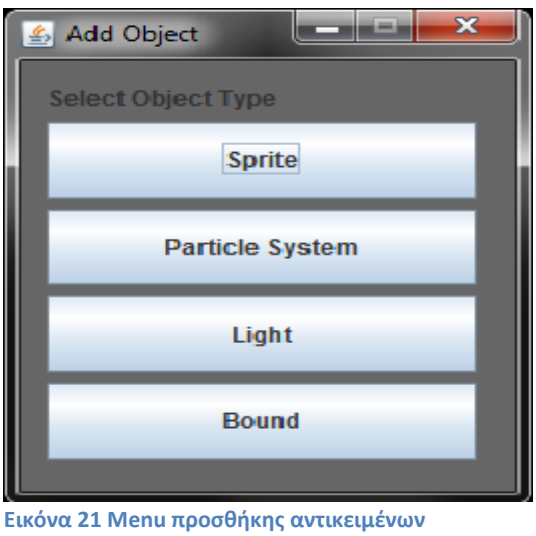
Εικόνα 20 Διεπαφή Scene Editor

1: Επιλογές για δημιουργία νέας, σώσιμο και φόρτωση υπάρχουσας σκηνής. Για να φορτώσουμε υπάρχουσα πρέπει να επιλέξουμε ολόκληρο τον φάκελο που την περιέχει.

2: Εργαλεία ζωγραφικής ίδια με αυτά που χρησιμοποιούνται στον Sprite Editor. Μία διαφορά είναι η δυνατότητα επιλογής μιας εικόνας για background της σκηνής με το κουμπί Load Background.

3: Το checkbox Paint Mode πρέπει να είναι επιλεγμένο άν θέλουμε να χρησιμοποιήσουμε τα εργαλεία ζωγραφικής.

4: Το πεδίο UPS ορίζει τα update ανά δευτερόλεπτο και με το κουμπί Start τρέχουμε την σκηνή. Όταν τρέχουμε την σκηνή απο τον Scene Editor δέν έχουμε έλενχο του χαρακτήρα και γενικά δέν τρέχει η λογική του παιχνιδιού αλλά μπορούμε να δούμε μόνο τα ενεργά animation και τους φωτισμούς ώστε να έχουμε μία καλύτερη εικόνα για το αποτέλεσμα πρήν το μεταφέρουμε στον κώδικα.



Εικόνα 21 Menu προσθήκης αντικειμένων

5: Με το κουμπί ΄+΄ ανοίγει το παρακάτω παράθυρο απο το οποίο μπορούμε να προσθέσουμε αντικείμενα στην σκηνή.

Sprite: Για γραφικά αντικείμενα. Επιλέγουμε ή αρχείο εικόνας (.png) , ή ολόκληρο αρχείο της μορφής Animated Sprite.

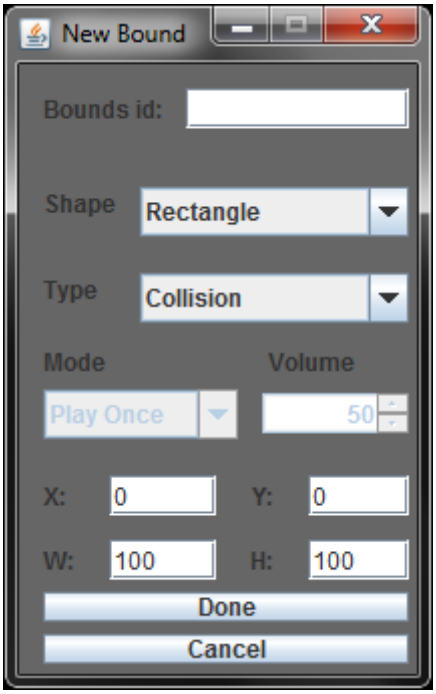
Particle System: Για συστήματα σωματιδίων.

Light: Εμφανίζει το παράθυρο πιο κάτω για να ορίζουμε φωτισμούς.

Bound: Εμφανίζει το παράθυρο πιο κάτω για να φτιάχνουμε Bound.

Με το κουμπί ΄-΄ αφαιρούμε οποιοδήποτε αντικείμενο έχουμε επιλεγμένο στην σκηνή.

Άν επιλέξουμε bound εμφανίζετε το παρακάτω παράθυρο:



Εικόνα 22 Menu δημιουργίας/επεξεργασίας Bound

Bounds id: Ανάλογα τι τύπος είναι το bound πρέπει να θέσουμε το κατάλληλο id. Για παράδειγμα άν είναι τύπου Sound το οποίο εκκινεί ήχο, το id του πρέπει να είναι ίδιο με το όνομα του ήχου. Άν είναι Entrance, δηλαδή αλλάζει την σκηνή πρέπει το id να είναι ίδιο με το όνομα αυτής της σκηνής. Για απλά τύπου Collision το id μπορεί να είναι ότι επιλέξουμε.

Shape: Ορίζει το σχήμα του bound (Rectangle - Ellipse)

Type: Τι είδους γεγονότα πυροδοτεί (Collision – Sound - Entrance)

Mode: Αν επιλέξουμε τύπου Sound μπορούμε να διαλέξουμε το mode (Play Once - Repeat) και την ένταση.

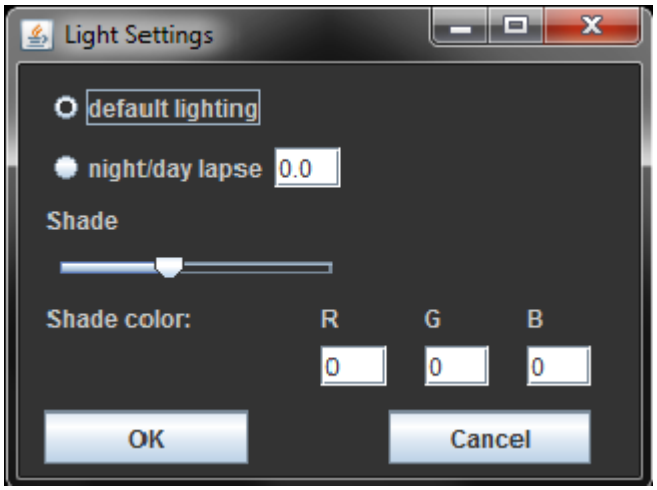
X,y,w,h: Οι συντεταγμένες και διαστάσεις του bound.

6: Drop box για να επιλέγουμε το επίπεδο (layer) της σκηνής στο οποίο θα προσθέτονται τα αντικείμενα.

7: Λίστα με όλα τα γραφικά αντικείμενα της σκηνής.

8: Λίστα με όλα τα συστήματα σωματιδίων της σκηνής.

9: Λίστα με όλους τους φωτισμούς της σκηνής. Το κουμπί Lighting Settings ανοίγει το παράθυρο για τις ρυθμίσεις φωτισμών της σκηνής.



Εικόνα 23 Επιλογές φωτισμού σκηνής

default lighting: Όταν είναι επιλεγμένο η σκηνή ζωγραφίζετε στα κανονικά της χρώματα και φωτεινότητα και όλοι οι φωτισμοί απενεργοποιούνται. **Για να έχουμε οποιοδήποτε εφέ φωτισμού πρέπει αυτό το πεδίο να είναι απενεργοποιημένο.**

night/day lapse: Όταν είναι επιλεγμένο ο φωτισμός της σκηνής θα αυξομειώνει αυτόματα για να προσομοιώσει την εναλλαγή μέρας - νύχτας με ρυθμό που ορίζουμε στο πεδίο αριστερά.

Shade : Το επίπεδο φωτισμού της σκηνής.

Shade color: ορίζουμε το χρώμα που θα σκιάζεται η σκηνή δίνοντας στα πεδία R G B τις τιμές κόκκινο , πράσινο και μπλέ αντίστοιχα.

10: Λίστα με όλα τα bound της σκηνής. Με το checkbox Render επιλέγουμε άν τα bound θα είναι ορατά.

11: Το check box Object Mode πρέπει να είναι ενεργό άν θέλουμε να κάνουμε οποιαδήποτε αλλαγή στα αντικείμενα.

4 Αποτελέσματα

Το αποτέλεσμα αυτής της πτυχιακής είναι ένα εργαλείο που μπορεί να εξυπηρετήσει στην δημιουργία ηλεκτρονικών παιχνιδιών, κάνοντας την διαδικασία πιο γρήγορη, οργανωμένη και ενδιαφέρουσα. Μειώνει σε μεγάλο βαθμό τον κώδικα που απαιτεί ένα παιχνίδι και προσφέρει πολλά έτοιμα αλλά παραμετροποιήσημα στοιχεία που χρειάζονται συχνώς σε αυτά. Δίνει την δυνατότητα στον χρήστη να προσπεράσει την προεργασία και τις λεπτομέρειες της υλοποίησης και να μπορεί να αφιερώσει περισσότερο χρόνο στα σημεία πού κάνουν ένα παιχνίδι ενδιαφέρον όπως πλοκή, τέχνη και εφέ.

Αυτή η εργασία μπορεί να ωφελήσει ανεξάρτητους (indie) δημιουργούς παιχνιδιών οι οποίοι συνήθως δουλεύουν σε μικρές ομάδες η ακόμη και ατομικά και στοχεύουν σε απλότητα και ποιότητα παρά την πολυπλοκότητα και την λεπτομέρεια που θα απαιτούσε μια μεγάλη ομάδα και κάποιο κεφάλαιο. Τέτοια παιχνίδια είδαν μεγάλη ανάπτυξη αυτά τα χρόνια. Λόγω της εύκολης και μεγάλης πρόσβασης σε πληροφορίες και γνώσεις πού μας προσφέρει το ίντερνετ, όλο και περισσότεροι άνθρωποι μαθαίνουν να προγραμματίζουν είτε γιατί ψάχνουν κάποιο χόμπι είτε γιατί κυνηγούν κάποιο επιπλέον εισόδημα. Πολλοί από αυτούς επιλέγουν να φτιάχνουν παιχνίδια και πλέον υπάρχει μια μεγάλη κοινότητα από τέτοιους δημιουργούς και πολλοί εξυπηρετητές που τους βοηθούν να ξεκινήσουν και προωθούν την δουλειά τους. Αυτή η εργασία θα βοηθήσει πολλούς οι οποίοι θέλουν μια γρήγορη και εύκολη αρχή χωρίς να εμβαθύνουν σε λεπτομέρειες.

Επιπλέον καθώς δίνουμε την ελευθερία στον χρήστη να δει και να επεξεργαστεί εύκολα τα αρχεία που δομούν ένα παιχνίδι, μπορεί να δει άμεσα τα αποτελέσματα και τις επιπτώσεις των αλλαγών μέσα σε αυτό, έτσι αυτή η εργασία μπορεί να χρησιμοποιηθεί και ως εφαρμογή εκμάθησης προγραμματισμού.

Εκτός από τους χρήστες της, αυτή η εργασία ωφέλησε και εμένα. Μου πρόσφερε γνώσεις από ένα εύρη πεδίο εφαρμογών, η δημιουργία παιχνιδιών απαιτεί την ενασχόληση με πολλά θέματα όπως γραφικά, ήχος, είσοδος-έξοδος, πολυεπεξεργασία, δικτύωση, συγχρονισμός, βελτιστοποίηση, φυσική κα. Επίσης αυτή η εργασία μου πρόσφερε μια πρώτη επαφή με την διαδικασία σχεδιασμού , υλοποίησης και αποσφαλμάτωσης ενός τέτοιου μεγέθους πρότζεκτ υπό χρονοδιάγραμμα.

4.1 Μελλοντική εργασία

Από την μεριά της βιβλιοθήκης (JIGL) μία ιδέα είναι η επέκταση σε Android, καθώς χρησιμοποιεί Java, με την σημερινή χρήση κινητών συσκευών και την ελεύθερη είσοδο στο Android market, θα είναι μια αναγκαία επέκταση. Μεγάλο μέρος της βιβλιοθήκης μπορεί ίδη να χρησιμοποιηθεί για αυτό το σκοπό αλλά πολλά στοιχεία της χρειάζονται προσαρμογή. Επίσης θα ήταν ενδιαφέρον ενα σύστημα τεχνητής νοημοσύνης που ο χρήστης θα μπορούσε εύκολα να ορίσει συμπεριφορές αντικειμένων βασισμένες σε συνθήκες. Ένα σύστημα σαν αυτό θα μείωνε κατα πολύ την ποσότητα κώδικα που θα χρειαζόταν το παιχνίδι και θα έφερνε αυτήν την εργασία πολύ πιο κοντά σε έναν απο τους κύριους στόχους της. Επίσης χρειάζεται ένα πιο ολοκληρωμένο σύστημα για multiplayer (διαδικτυακό παιχνίδι) . Αν και υπάρχουν ίδη κλάσεις για την δημιουργία και σύνδεση σε εικονικά δωμάτια υπάρχει ανάγκη και για άλλες λειτουργίες, όπως διανομή τυχαίων αριθμών για τυχαία γεγονότα στο παιχνίδι, καλύτερο πρωτόκολλο επικοινωνίας ώστε να συνδυάζει δεδομένα παιχνιδιού με επικοινωνία παιχτών, λειτουργία επανασύνδεσης σε περίπτωση σφάλματος κλπ. Ένα σημαντικό μέρος που δεν υλοποιήθηκε λόγω χρόνου είναι μια μηχανή φυσικής. Και αυτός είναι ο λόγος που η βιβλιοθήκη δεν μπορεί ακόμα να διαχειριστεί συμβάντα συγκρούσεων από μόνη της. Αυτή η υλοποίηση θα είναι το επόμενο βήμα στην εξέλιξη αυτής της εργασίας.

Καθώς αυτή η εργασία μεγαλώνει και τα αντικείμενα και τα χαρακτηριστικά τους πληθαίνουν, θα χρειαστεί και μια πιο ολοκληρωμένη γλώσσα για την περιγραφή τους, αλλά συγχρόνως να παραμείνει απλή στην χρήση ώστε να μην αντισταθμίσει τον κώδικα που θα μειώσει.

Μία άλλη εναλλακτική λύση είναι να επεκταθεί η βιβλιοθήκη ώστε να χρησιμοποιεί μια γλώσσα scripting όπως η Groovy. Έχοντας την δυνατότητα ο χρήστης να γράφει script και να τα προσαρμόζει εύκολα στα αντικείμενα του παιχνιδιού θα του δώσει μεγαλύτερη ελευθερία και επιλογές όχι μόνο στην δομή αλλά και στην λογική του παιχνιδιού.

Από την μεριά του editor (GCE) μία πρώτη αλλαγή θα ήταν η μετατροπή του σε environment, να έχει καλύτερη οργάνωση και διαχείριση των πρότζεκτ και με περισσότερους sub-editor ώστε να γίνει πιο εύκολη η δημιουργία και η διαχείριση των αντικειμένων. Επιπλέον θα ήταν χρήσιμο η δημιουργία repository για ομαδική δουλειά εξ'αποστάσεως Δηλαδή δημιουργία server όπου φυλάσσετε η ομαδική εργασία και τα μέλη συνδέονται για την λήψη, επεξεργασία και ενημέρωση των αρχείων της. Θα περιλαμβάνει διάφορες λειτουργίες όπως δικαιώματα αρχείων, τηλεδιάσκεψη, χρονοδιάγραμμα πρότζεκτ κα. Τέλος καθώς η βιβλιοθήκη προσπαθεί να κάνει εύκολη την δημιουργία παιχνιδιών, πρέπει και ο editor να αποκτήσει μία πιό εύχρηστη και φιλική με τον χρήστη διεπαφή. Μια τέτοια αλλαγή θα αύξανε την παραγωγικότητα και θα απλοποιούσε κάλο την διαδικασία. Δυστυχώς λόγο του μεγέθους αυτής της εργασίας και του περιορισμένου χρόνου δεν έδωσα τόση έμφαση στην αισθητική και την οργάνωση αλλά περισσότερο στις πιο βασικές και χρήσιμες λειτουργίες.

Bibliography

Davison, A. (n.d.). *Killer Game Programming In Java.*
Harold, E. R. (n.d.). *Java Network Programming 3rd Edition.*
Sarah Frisken, R. P. (2002). Simple and Efficient Traversal Methods for Quadtrees and Octrees.
Rogers Cadenhead, Laura Lemay Πλήρες εγχειρίδιο της Java 6

Παράρτημα

Γειά σας, ονομάζομαι Κάντι Βίκτωρας, κατάγομαι απο το ηράκλειο και είμαι στο ΠΕ εξάμηνο σπουδών. Αυτή η πτυχιακή έχει τίτλο ‘’Δημιουργία δισδιάστατων παιχνιδιών σε Java‘’ και στοχεύει στην απλοποίηση της κατασκευής τους εισάγοντας ένα νέο τρόπο περιγραφής και δομής τους.

Με την μεγάλη ανάπτυξη της βιομηχανίας παιχνιδιών, πολλοί πίστευαν πως η γενιά των δισδιάστατων παιχνιδιών θα εκλείψει. Αντίθετα όμως με την σημερινή πρόσβαση σε πληροφορίες, οδηγούς και παραδείγματα που προσφέρει το ίντερνετ πολλοί άνθρωποι μαθαίνουν να προγραμματίζουν είτε γιατί ψάχνουν κάποιο χόμπι είτε γιατί ψάχνουν ένα επιπλέον εισόδημα, και πολλοί από αυτούς επιλέγουν τα ηλεκτρονικά παιχνίδια. Σε τέτοιο βαθμό που πλέον υπάρχουν πολλοί εξυπηρετητές όπως το Steam, Itch, Desura, IndieCity όπως επίσης και το Android Market και το Mac Store για φορητές συσκευές που χρησιμοποιούνται όλο και περισσότερο πλέον, οι οποίοι βοηθούν τους ερασιτέχνες δημιουργούς να ξεκινήσουν και να διανέμουν τη δουλεία τους. Όμως η δημιουργία ηλεκτρονικών παιχνιδιών είναι μια διαδικασία αρκετά δύσκολη και χρονοβόρα ειδικά για ερασιτέχνες που δουλεύουν μόνοι η σε μικρές ομάδες. Απαιτεί γνώσεις απο πολλά πεδία και πολύ προεργασία πρίν δούμε κάποιο αποτέλεσμα. Αυτή η εργασία έχει στόχο να κρύψει τις λεπτομέρειες της υλοποίησης, μειώνοντας όσο το δυνατόν τον κώδικα που απαιτεί ενα παιχνίδι και να δώσει στον δημιουργό περισσότερο χρόνο για τα σημεία που κάνουν ένα παιχνίδι ενδιαφέρον όπως τέχνη, πλοκή και εφέ.

Κριτήριο για την επιλογή αυτής της εργασίας ήταν πρώτον η ποικιλία των θεμάτων που θα έπρεπε να ασχοληθώ, τα ηλεκτρονικά παιχνίδια περιλαμβάνουν την υλοποίηση πολλών δομών και αλγορίθμων και την αντιμετώπιση διάφορων προβλημάτων. Δεύτερον η επεκτασιμότητα, ήθελα να μήν εχω περιορισμό στο πόσα πράγματα μπορώ να δοκιμάσω γύρω απο το ίδιο θέμα και ήθελα να ξεκινήσω μια εργασία της οποίας η ανάπτυξη δέν θα τελείωνε με την πτυχιακή.

Όπως θα δείτε από την αναφορά είναι μία πτυχιακή η ο οποία ασχολείται με πολλά θέματα. Από διαχείριση αρχείων, είσοδο χρήστη και συγκρούσεις μέχρι animation, rendering, φωτισμό και ήχο. Επίσης στην αναφορά παρέλειψα κάποια θέματα λόγω μικρής σημασίας, χρόνου και επειδή προτίμησα να αναφερθώ σε άλλα που δεν έχουν υλοποιηθεί ακόμα σαυτήν την εργασία αλλά θεώρησα πιο σημαντικά όπως Ray casting, Procedural generation και 3D.

Η εργασία χωρίζετε σε δύο μέρη. Ένα περιβάλλον (GCE) στο οποίο μπορούμε να παράγουμε περιεχόμενο για τα παιχνίδια, όπως animated χαρακτήρες, φωτισμούς, σωματίδια, ήχους, σκηνές. Και μία βιβλιοθήκη η οποία μπορεί να φορτώσει άμεσα αυτά τα αντικείμενα και παρέχει συναρτήσεις για να τα χρησιμοποιήσουμε. Αντικείμενα τα οποία μπορούν να χρησιμοποιηθούν είτε αυτόνομα είτε μαζί για να δομήσουν μια διαδραστική σκηνή και τα οποία μπορούν να προσαρμοστούν εύκολα σε κάθε περίπτωση χρήσης και τύπο παιχνιδιού. Αυτό το περιεχόμενο περιγράφετε απο αρχεία κειμένου τα οποία μπορούν εύκολα να τροποποιηθούν και χειροκίνητα με αποτέλεσμα να μπορούμε να κάνουμε αλλαγές στο παιχνίδι χωρίς να πειράξουμε τον κώδικα.

Το περιβάλλον χωρίζεται σε τρείς βασικούς editor. Έναν για να φτιάχνουμε συστήματα σωματιδίων, ο οποίος μας δίνει πολλές επιλογές για να καθορίσουμε τα χαρακτηριστικά και την συμπεριφορά του συστήματος όπως ταχύτητα, επιτάχυνση ,κατεύθυνση, δυνάμεις, εύρος ζωής, χρώμα. Με αυτόν τον editor μπορούμε να δημιουργούμε εφέ όπως βροχή, χιόνι, φωτιά, καπνό και γενικά αντικείμενα χωρίς προκαθορισμένο σχήμα. Ο δεύτερος editor χρησιμοποιείτε για να φτιάχνουμε sprites. Περιλαμβάνει βασικά εργαλεία ζωγραφικής για να φτιάχνουμε καρέ και μπορούμε να ορίσουμε σειρές απο τέτοια καρέ ως animation. Και ο τρίτος editor χρησιμοποιείτε για να δέσουμε αυτα τα αντικείμενα σε μία σκηνή και να προσθέσουμε ήχους, φωτισμούς και bounds.

Αφού φτιάξουμε μια σκηνή μπορούμε να την φορτώσουμε και να την τρέξουμε χρησιμοποιώντας την βιβλιοθήκη η οποία μας δίνει και την δυνατότητα να ενσωματώσουμε εύκολα την λογική πίσω από αυτήν. Η βιβλιοθήκη επίσης περιέχει πολλά άλλα αντικείμενα και αλγόριθμους που χρησιμοποιούνται συχνά έως πάντα στην κατασκευή παιχνιδιών όπως display, camera, quadtrees, diamond square, culling, διαχείρισης ουδέτερων αντικειμένων, βασικές κλάσεις για networking και αρκετά άλλα που μπορούν να συνδυαστούν με τα αντικείμενα της σκηνής για να παράγουν διαδραστικό περιεχόμενο.

Το αποτέλεσμα αυτής της εργασίας είναι ένα εργαλείο που απλοποιεί σε μεγάλο βαθμό την δημιουργία ηλεκτρονικών παιχνιδιών, κάνει την διαδικασία πιο δημιουργική, πιο οργανωμένη και μειώνει σημαντικά τον κώδικα που απαιτείτε. Χρήσιμο για ερασιτέχνες (indie) δημιουργούς που δουλεύουν μόνοι η σε μικρές ομάδες και στοχεύουν σε απλότητα και ποιότητα παρά την πολυπλοκότητα και την λεπτομέρεια που θα απαιτούσε μια μεγάλη ομάδα και κάποιο κεφάλαιο. Επιπλέον δίνοντας την ελευθερία στον προγραμματιστή να πειράζει μεταβλητές και να βλέπει άμεσα τις επιπτώσεις μέσα στο παιχνίδι αυτή η εργασία θα μπορούσε να χρησιμοποιηθεί και για εκπαιδευτικούς σκοπούς.

Κάποια μέρη αυτής της βιβλιοθήκης δέν είναι ολοκληρωμένα ή βρίσκονται σε αρχικό στάδιο όπως η δικτύωση, τεχνητή νοημοσύνη, φυσική και 3d αλλα είναι τα πρώτα βήματα για την μελλοντική επέκταση της. Απο την μεριά της βιβλιοθήκης μία πρώτη ιδέα είναι η επέκταση σε Android, καθώς όλη η εργασία είναι γραμμένη σε Java. Μεγάλο μέρος της βιβλιοθήκης μπορεί ίδη να χρησιμοποιηθεί για αυτο τον σκοπό αλλα και πολλά αντικείμενα χρειάζονται προσαρμογή. Επίσης όσο αυτή η εργασία μεγαλώνει και τα αντικείμενα και τα χαρακτηριστικά τους πληθαίνουν θα χρειαστεί και μια πιό ολοκληρωμένη γλώσσα για την περιγραφή τους άλλα που θα παραμείνει κατανοητή και εύκολη στην χρήση ώστε να μην αντισταθμίσει τον κώδικα που θα γλυτώσει. Μία εναλλακτική θα ήταν η ενσωμάτωση μιας γλώσσας scripting όπως η Groovy η οποία θα έδινε μεγαλύτερη ευελιξία στην υλοποίηση της λογικής του παιχνιδιού καθώς ακόμη η εργασία συγκεντρώνεται πιό πολύ στην δομή. Τέλος κάτι σημαντικό που λείπει απο την βιβλιοθήκη είναι μία μηχανή φυσικής η οποία χρησιμεύει σχεδόν σε κάθε παιχνίδι και γλιτώνει μεγάλο μέρος της υλοποίησης. Από την μεριά του GCE μια βελτίωση θα ήταν η μετατροπή του σε περιβάλλον παρά σε editor που μοιάζει τώρα, με καλύτερη διαχείριση των πρότζεκτ, περισσότερους sub-editors, και πιο φιλική διεπαφή. Επιπλέον θα μπορούσε να περιλαμβάνει την δυνατότητα online repository για ομαδική δουλειά μέσω δικτύου. Δηλαδή να δημιουργείται server που θα φιλάει όλα τα αρχεία και τα μέλη της ομάδας θα συνδέονται εκεί για την λήψη, επεξεργασία και ενημέρωση τους.