ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ

Σχολή Τεχνολογικών Εφαρμογών

Τμήμα Εφαρμοσμένης Πληροφορικής και Πολυμέσων

Πτυχιακή Εργασία

«Επέκταση μηχανής 3D γραφικών για υποστήριξη πεδίων μεγάλης έκτασης»

ΟΝΟΜΑΤΕΠΩΝΥΜΟ: Σταυρακάκης Ιωάννης

Σταυρακάκης Νικόλαος

Ημερομηνία: 10 / 10 / 2014

ΕΙΣΗΓΗΤΗΣ: Παπαδάκης Χαράλαμπος

Σε όσους βοήθησαν και στήριξαν τις προσπάθειες μας,

με ιδιαίτερη εκτίμηση!

Ευχαριστίες

Με την ολοκλήρωση της πτυχιακής εργασίας, η οποία υλοποιήθηκε υπό την επίβλεψη του κ. Παπαδάκη Χαράλαμπου, θα θέλαμε να ευχαριστήσουμε όλους εκείνους οι οποίοι πρόσφεραν πολύτιμη βοήθεια για την περάτωση αυτής της εργασίας. Θα ήταν παράλειψη να μην αναφερθώ σε όλους εκείνους που συμπαραστάθηκαν σε αυτήν την προσπάθεια. Κατά κύριο λόγω, πρέπει να ευχαριστήσουμε τον επιβλέποντα καθηγητή από το Τ.Ε.Ι. Κρήτης κ. Χαράλαμπο Παπαδάκη, ο οποίος μας υποστήριξε καθ' όλη τη διάρκεια της πτυχιακής εργασίας καθώς και το προσωπικό μου φίλο Κατσαρό Δημήτριο για τις συνεχείς συμβουλές που μας παρείχε. Στη συνέχεια, τους γονείς μας για την στήριξη την οποία προσέφεραν με όποιον τρόπο μπορούσαν και όλους εκείνους στους οποίους βρήκαμε στήριγμα είτε ψυχολογικά είτε υλικά..

Ηράκλειο, Οκτώβριος 2014

Σταυρακάκης Ιωάννης
Σταυρακάκης Νικόλαος

Περιεχόμενα

Πίνακας περιεχομένων

Κεφάλαιο Ι

Περίληψη Πτυχιακής Εργασίας

Η πτυχιακή εργασία αυτή προσφέρει τη δυνατότητα στον χρήστη να δημιουργήσει ένα τρισδιάστατο τοπίο (τυχαίο ή από το μηδέν), να το επεξεργαστεί όπως επιθυμεί και τελικά να το αποθηκεύσει. Δίνει επίσης τη δυνατότητα να προστεθεί ήλιος, ουρανός, θάλασσα καθώς και διαφόρων αντικειμένων πάνω στο τοπίο όπως πέτρες, δέντρα, χιόνι κλπ.. Με άλλα λόγια, μπορούμε να δημιουργήσουμε ένα δικό μας κόσμο. Υπάρχει η δυνατότητα πλοήγησης μέσα στο κόσμο προς όλες τις κατευθύνσεις.

Το λογισμικό που αναπτύχθηκε στα πλαίσια της εργασίας μπορεί να χρησιμοποιηθεί μετέπειτα ως βάση για τη δημιουργία κάποιου παιχνιδιού. Όσοι ασχολούνται με τη δημιουργία παιχνιδιών γνωρίζουν ότι η δημιουργία του τοπίου είναι βασικό συστατικό και δίνεται πολύ μεγάλη έμφαση και προσοχή στη κατασκευή του.

Η δυσκολία της εργασίας αυτής έγκειται στη δυναμικότητα την οποία προσφέρει διότι δεν μένουμε στην απλή δημιουργία ενός μεγάλου πεδίου αλλά στη δημιουργία μιας τεχνικής η οποία θα δημιουργεί ανάλογα με τις απαιτήσεις του χρήστη τοπία είτε αυτόματα είτε όχι, πάντα διαφορετικά. Μέσω μιας "γεννήτριας υψομετρικών εικόνων" δημιουργούνται ασπρόμαυρες εικόνες οι οποίες ουσιαστικά δείχνουν το ύψος σε κάθε σημείο στη συνέχεια τις επεξεργαζόμαστε μέσω κώδικα σε Java και τέλος τις εισάγουμε στο Visual Studio και στον Editor. Παρακάτω ακολουθεί επεξήγηση κάθε βήματος αναλυτικά.

Τα προγράμματα τα οποία χρησιμοποιήσαμε στην εργασία αυτή είναι τα εξής:

- Microsoft Visual Studio
- Esenthel Engine
- NetBeans
- L3DT Terrain Generator

Summary

This thesis offers the user the ability to create a three-dimensional landscape (random or from the begging), edit it as desired and eventually store it. It also allows to add sun, sky, sea and various objects on the landscape such as rocks, trees, snow, etc. .. In other words, we can create our own world. There is the ability to navigate in the world in all directions.

The software developed in this thesis can be used later as the basis for creating a game. Those involved in making games, know that the creation of the landscape is a key component and is given too much emphasis and attention.

The difficulty of this work lies in the capacity that offers because we do not live in the mere creation of a large field, but create a technique that creates, depending on user requirements, landscapes automatically or not, always different. Through an "elevation images generator" creates black and white images which show essentially the height at any point. After that, follows the process through code in Java and then import them into Visual Studio and Editor. Below is an explanation of each step in detail.

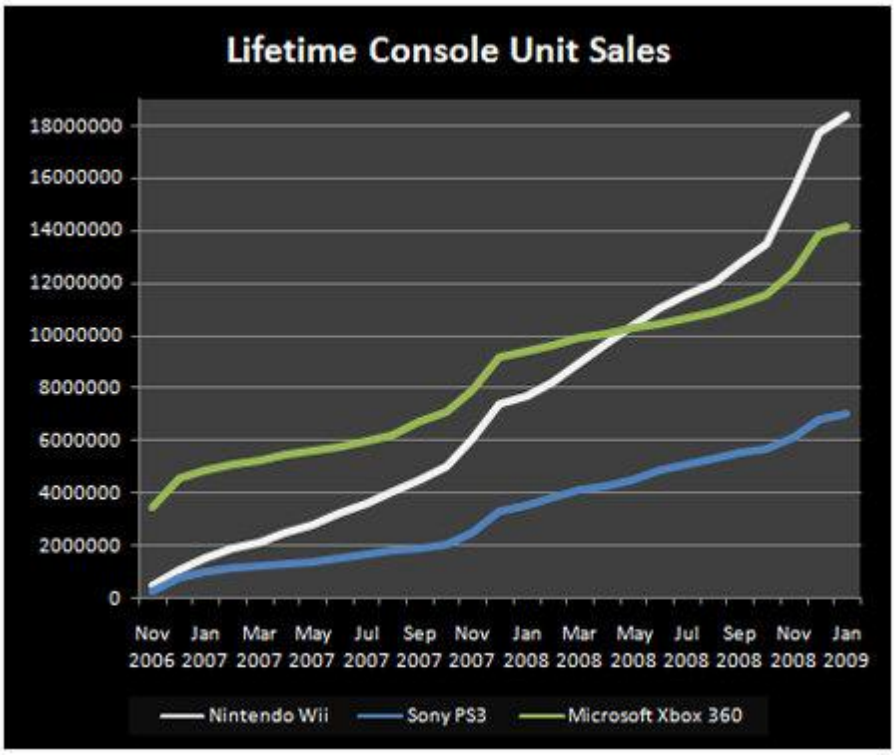
The programs used in this work are as follows:

- Microsoft Visual Studio
- Esenthel Engine
- NetBeans
- L3DT Terrain Generator

Κεφάλαιο ΙΙ

Ιστορική Αναδρομή

Από το 1971, όταν πρωτοεμφανίστηκε το πρώτο παιχνίδι από την εταιρία Atari μέχρι σήμερα, τα πράγματα έχουν αλλάξει πολύ στο τομέα της ανάπτυξης ηλεκτρονικών παιχνιδιών. Η έκρηξη ενδιαφέροντος από το κοινό για τις κονσόλες παιχνιδιών και για τα ίδια τα παιχνίδια είναι εκπληκτική. Πέρα από άλλες μορφές ψυχαγωγίας όπως είναι ο κινηματογράφος και άλλα, η στροφή ενδιαφέροντος του κόσμου για ψυχαγωγία στο σπίτι παρουσιάζει μεγάλο ενδιαφέρον. Ενδεικτικά παρακάτω παρατίθεται ένα διάγραμμα που δείχνει την αύξηση στις πωλήσεις κονσόλων παιχνιδιών.



Εικόνα 1: Αύξηση στις πωλήσεις κονσόλων, από τον Νοέμβριο του 2006 έως τον Ιανουάριο του 2009

Πηγή: www.tgdaily.com

Τα παιχνίδια τα οποία συγκαταλέγονται στην κατηγορία των προγραμμάτων, κατέχουν πλέον ένα σημαντικό κομμάτι της πίτας στην ψηφιακή αγορά. Αρχικά βλέπουμε να εμφανίζονται παιχνίδια ενός παίκτη, χωρίς μουσική και ιδιαίτερο σενάριο. Σήμερα έχουμε φτάσει στην εποχή όπου τα παιχνίδια διαθέτουν κινηματογραφικά εφέ, "χολιγουντιανά" σενάρια, και μπορούν να συμμετέχουν σ' αυτά παίκτες από όλο τον κόσμο. Στο τελευταίο βέβαια, σημαντικό ρόλο έχει παίξει και η ραγδαία ανάπτυξη του διαδικτύου τα τελευταία χρόνια. Πώς όμως φτάσαμε μέχρι εδώ;

Πηγαίνοντας πίσω στο παρελθόν, η εταιρία ανάπτυξης παιχνιδιών, η γνωστή σε όλους μας Atari, ξεκίνησε το 1972 την μαζική παραγωγή του παιχνιδιού Pong, το οποίο είχε μορφή κερματοδέκτη. Συνολικά πούλησε 38.000 τέτοιους κερματοδέκτες και το παιχνίδι ήταν ολοκληρωτικά σχεδιασμένο με κυκλώματα TTL, χωρίς καν να χρησιμοποιεί επεξεργαστή, ή κάποιο κώδικα (πρόγραμμα). Ο κώδικας του παιχνιδιού αυτού δημιουργήθηκε από βασικά δομικά στοιχεία ψηφιακών

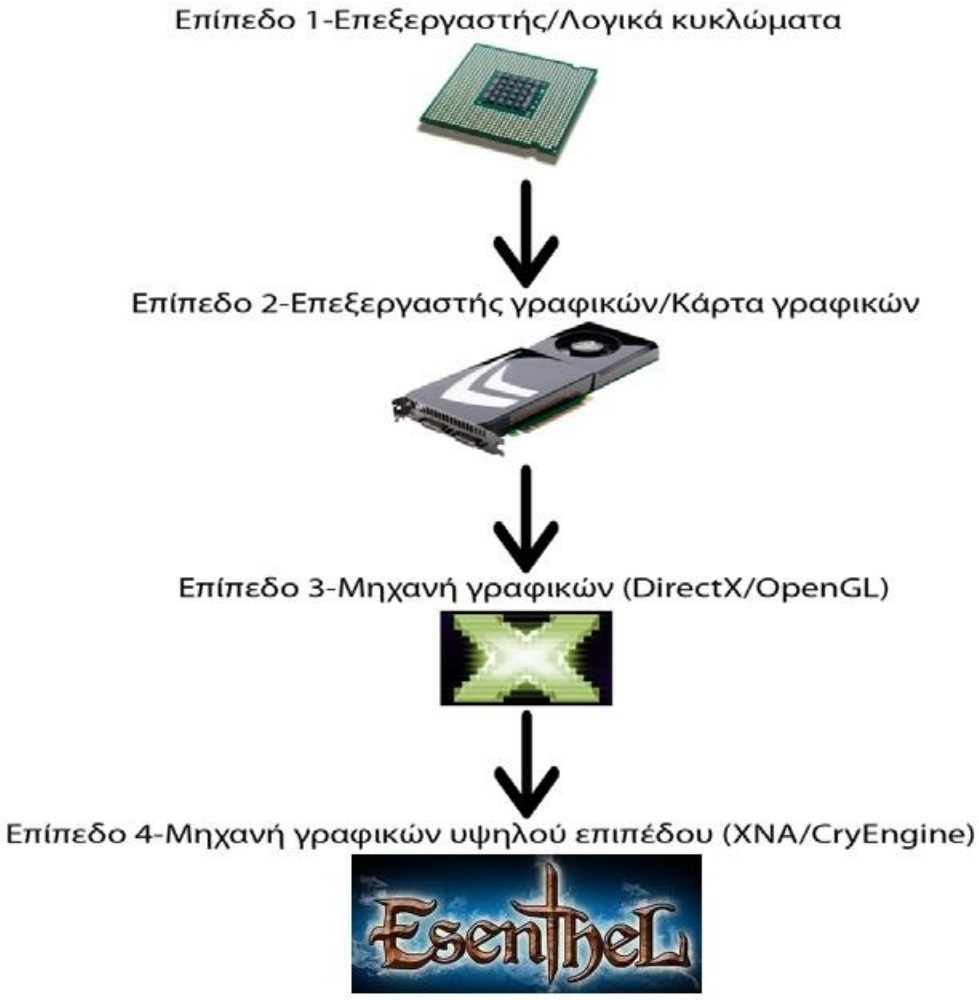
κυκλωμάτων (πύλες), μετρητές και χρονοδιακόπτες. Ήταν η γέννηση της αυτοκρατορίας των παιχνιδιών.

Εικόνα 2: Ο κερματοδέκτης της Atari για το παιχνίδι Pong

Λόγω του υψηλού ενδιαφέροντος του κοινού, οι εταιρίες ξεκίνησαν έναν ξέφρενο αγώνα παραγωγής παιχνιδιών από τότε. Λόγω του μεγάλου ανταγωνισμού και της αύξησης των απαιτήσεων από τη μεριά των χρηστών, τα παιχνίδια σιγά σιγά άρχισαν να γίνονται ολοένα και πιο περίπλοκα. Πλέον η χρήση βασικών ψηφιακών κυκλωμάτων για τη δημιουργία παιχνιδιών ήταν ανεπαρκής, λόγω του υψηλού κόστους, της περιπλοκότητας και της έλλειψης δυνατότητας επαναχρησιμοποίησης. Έτσι σταδιακά, και ακόμη πιο έντονα με την εμφάνιση του δομημένου προγραμματισμού οι εταιρίες ξεκίνησαν να αναπτύσσουν τα παιχνίδια τους σε κώδικα παρά σε ψηφιακά κυκλώματα. Αυτό αρχικά, κυρίως λόγω των τότε χαμηλών απαιτήσεων φαινόταν επαρκές. Όμως εξαιτίας της αύξησης της περιπλοκότητας των παιχνιδιών, άρχισαν να εμφανίζονται χιλιάδες γραμμές κώδικα οι οποίες ήταν δύσκολες στη συντήρησή τους και ακόμη πιο δύσκολο για κάποιον καινούριο προγραμματιστή στον χώρο των παιχνιδιών να τις κατανοήσει. Επίσης, αξίζει να σημειώσουμε εδώ, ότι πλέον ο επεξεργαστής γραφικών αποτελούσε ξεχωριστό κομμάτι του υπολογιστή, επειδή ο όγκος για την επεξεργασία δεδομένων ήταν πολύς και υπήρχε μεγάλη και άσκοπη σπατάλη πόρων του υπολογιστή και του επεξεργαστή, ειδικά σε συστήματα που έτρεχε από πίσω κάποιο λειτουργικό σύστημα. Το chip αυτό έγινε γνωστό με το όνομα **κάρτα γραφικών** από την εταιρία Microsoft στις 30 Σεπτεμβρίου 1995 με το όνομα **DirectX.**

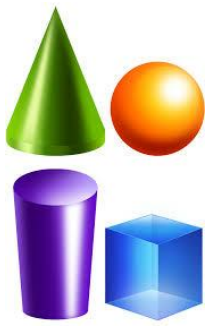
Με απλά λόγια, μια μηχανή γραφικών δεν είναι τίποτα άλλο παρά ομαδοποιημένες έτοιμες συναρτήσεις με βασικά εργαλεία που είναι χρήσιμα στην δημιουργία παιχνιδιών. Για παράδειγμα,

περιλαμβάνουν τον σχεδιασμό τριγώνων, τον έλεγχο συγκρούσεων, την δικτύωση και άλλα πράγματα τα οποία είναι κοινά για όλα τα παιχνίδια και δεν χρειάζεται ξανά και ξανά να τα εφευρίσκουμε από την αρχή. Σήμερα το DirectX έχει φτάσει αισίως στην έκδοση 11. Ο προγραμματισμός των παιχνιδιών έγινε ακόμα ευκολότερος και έχουμε την εμφάνιση ακόμη πιο εξεζητημένων και περίπλοκων παιχνιδιών. Με την ξέφρενη ανάπτυξη των παιχνιδιών και την εισαγωγή καινούριων εννοιών στο χώρο των βιντεοπαιχνιδιών, όπως για παράδειγμα η **Τεχνητή Νοημοσύνη**, η χρήση της μηχανής γραφικών DirectX ήταν πλέον δύσκολη, διότι περιελάμβανε έννοιες τριγώνων και σχημάτων και όχι κάτι πιο εξειδικευμένο. Έτσι εφευρέθηκαν νέες μηχανές γραφικών οι οποίες ομαδοποίησαν τις εντολές DirectX σε ένα επίπεδο πιο επάνω. Σε ακριβώς αυτό το επίπεδο βρίσκεται και η μηχανή γραφικών που χρησιμοποιείται σε αυτή την πτυχιακή εργασία, η μηχανή **Esenthel** άρρηκτα συνδεδεμένη με το **Microsoft Visual Studio** (**C++**). Άλλες παρόμοιες μηχανές ίδιου επιπέδου είναι η **Cry Engine** από την εταιρία CryTek και η **Unreal Engine**, από την εταιρία Unreal Technologies. Ένα διάγραμμα για να κατανοήσουμε τα επίπεδα αυτά ακολουθεί παρακάτω:



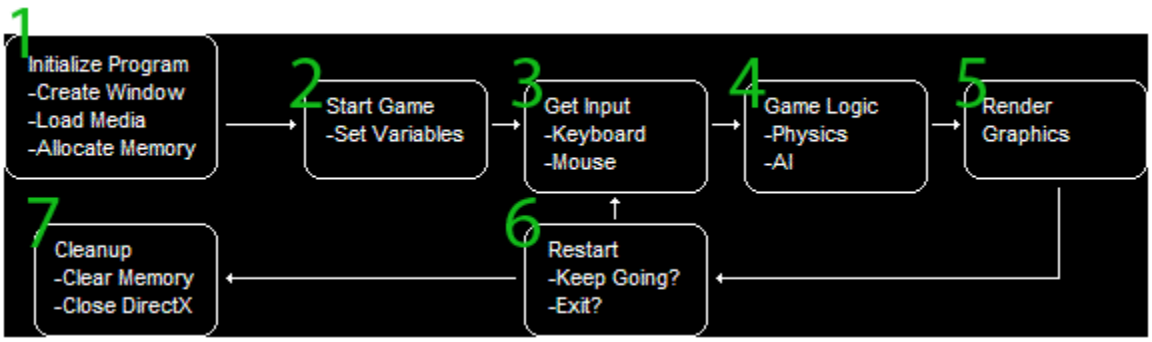
Εικόνα 3: Τα επίπεδα προγραμματισμού γραφικών

Κεφάλαιο ΙΙΙ



Εισαγωγή στον Προγραμματισμό Τρισδιάστατων Γραφικών

Όπως είδαμε σε προηγούμενο κεφάλαιο, ο προγραμματισμός των παιχνιδιών λόγω των υψηλών απαιτήσεων και της αύξησης περιπλοκότητας γίνεται ολοένα και με πιο εξελιγμένα εργαλεία. Παρ' όλη όμως την χρήση των εξελιγμένων εργαλείων αυτών, όλες οι μηχανές γραφικών και τα εξελιγμένα προγράμματα που χρησιμοποιούνται για την σχεδίαση μοντέλων, όπως είδαμε, στην βάση τους χρησιμοποιούν την ίδια λογική. Πριν ξεκινήσουμε τον σχεδιασμό μοντέλων και άλλων όμορφων πραγμάτων με την χρήση μιας μηχανής γραφικών πρέπει να κατασκευάσουμε έναν «χώρο» μέσα στον οποίο θα μπορούμε να εμφανίσουμε τα μοντέλα μας. Η εμφάνιση μοντέλων, τριγώνων και άλλων αντικειμένων που χρησιμοποιούνται στα παιχνίδια γίνεται μέσα σε ένα παράθυρο. Αυτή η διαδικασία εμφάνισης αντικειμένων στην οθόνη μέσω του παραθύρου αναφέρεται συχνά στον κόσμο των παιχνιδιών με την Αγγλική ορολογία **render**, ο οποίος και θα χρησιμοποιείται συχνά από εδώ και πέρα. Η μετάφραση στα Ελληνικά δεν είναι και τόσο εύστοχη και για αυτό το λόγο αυτός ο όρος, όπως και πολλοί άλλοι μέσα σε αυτήν την εργασία, θα αναφέρονται με την Αγγλική ορολογία, γιατί πολλές φορές η μετάφραση από τα Αγγλικά στα Ελληνικά είναι άστοχη**.** Προχωρώντας λοιπόν και με ελάχιστη γνώση προγραμματισμού, για να σχεδιάσουμε γραφικά γίνεται κατανοητό πως πρέπει πρωτίστως να σχεδιάσουμε το παράθυρό μας με μια γλώσσα προγραμματισμού, όπως για παράδειγμα με την C# ή την C++, η οποία και χρησιμοποιείται σε αυτήν την εργασία. Τα «έτοιμα» παράθυρα που μας προσφέρουν τα σύγχρονα περιβάλλοντα ανάπτυξης εφαρμογών όπως το **Visual Studio** της **Microsoft**, δεν μας είναι αρκετά γιατί θα θέλαμε ιδανικά να έχουμε τον πλήρη έλεγχο του παραθύρου και να χειριζόμαστε τα **interrupts** (τα σήματα διακοπής που δέχεται ο Η/Υ ώστε να επικοινωνήσει με συσκευές εισόδου) τα οποία αυτό λαμβάνει, όπως για παράδειγμα την κίνηση του ποντικιού, το πάτημα ενός πλήκτρου στο πληκτρολόγιό μας και άλλα. Στη δική μας περίπτωση εισάγοντας τις έτοιμες βιβλιοθήκες της μηχανής **Esenthel** στο Visual Studio αυτόματα γλυτώνουμε πολύ κόπο και χρόνο για ορισμένα εκ των παραπάνω. Στην ουσία, όταν αναπτύσσουμε εφαρμογές με τα σύγχρονα περιβάλλοντα ανάπτυξης, αυτά όλα κατασκευάζονται αυτόματα με κώδικα. Πρέπει, επίσης, να έχουμε υπόψη μας ότι τα παράθυρα αυτά, δεν είναι στατικά, δηλαδή όταν «τρέχουμε» το πρόγραμμα και βλέπουμε το παράθυρο αυτό δεν «περιμένει» θα λέγαμε από μας να πατήσουμε ένα πλήκτρο για να κάνει κάτι, αλλά τρέχοντάς το πρόγραμμα μπαίνει σε έναν ατέρμονο βρόγχο και σε κάθε επανάληψη ελέγχει για τυχόν **εισόδους** από τον χρήστη, δηλαδή για πάτημα πλήκτρου, κίνηση ποντικιού και άλλα. Αυτός ακριβώς ο βρόγχος είναι ζωτικής σημασίας για το παιχνίδι μας και είναι κάτι το οποίο μαθαίνει κάποιος νέος προγραμματιστής εισερχόμενος στον χώρο των βιντεοπαιχνιδιών. Ακόμη, αυτός ο βρόγχος είναι γνωστός και αναφέρεται στον προγραμματισμό παιχνιδιών με τον όρο **Main Game Loop**. Γενικά για κατανοήσουμε τα βασικά και να δούμε με ποια λογική «τρέχει» ένα παιχνίδι, παρακάτω παρατίθεται ένα διάγραμμα που θα μας βοηθήσει να κατανοήσουμε τις **εφτά φάσεις ενός παιχνιδιού.**



Εικόνα 4: Οι εφτά φάσεις ενός παιχνιδιού
Πηγή: www.directxtutorial.com

Όπως αναφέραμε και στην αρχή αυτού του κεφαλαίου και βλέπουμε στην εικόνα 4, πριν ακόμα ξεκινήσουμε να σχεδιάζουμε μοντέλα και άλλα πράγματα με την μηχανή γραφικών, πρέπει να περάσουμε από την φάση 1 η οποία περιλαμβάνει κατασκευή ενός παραθύρου και τη δέσμευση της απαραίτητης μνήμης.

Φάση 1- Κατασκευή παραθύρου, δέσμευση μνήμης

Από εδώ ακριβώς ξεκινάει το παιχνίδι μας. Γίνεται η φόρτωση της μηχανής γραφικών, όπως για παράδειγμα η **DirectX,** η κατασκευή του παραθύρου και η παραμετροποίησή του, δηλαδή εάν είναι πλήρη οθόνη, τι τίτλο θα έχει και άλλα. Επίσης, γίνεται ο ορισμός του παραθύρου αυτού ως το **main** παράθυρο και ότι ο χειριστής του θα είναι η μηχανή γραφικών που έχουμε ορίσει. Με απλά λόγια ρυθμίζουμε την μηχανή γραφικών να «ζωγραφίζει» μέσα σε αυτό το παράθυρο. Τέλος, γίνεται η απαραίτητη δέσμευση μνήμης που θα χρειαστούμε.

Φάση 2- Εκκίνηση παιχνιδιού

Εδώ πλέον φτάνουμε ένα βήμα πριν την **Main Game Loop** όπως αναφέραμε και πριν. Είναι το σημείο εκκίνησης του ατέρμονου βρόγχου, από τον οποίο και δεν θα «βγει» το παιχνίδι μας μέχρι να το σταματήσουμε εμείς. Τον τρόπο θα τον αναφέρουμε στην συνέχεια. Σε αυτήν την φάση, φορτώνουμε τα μοντέλα μας, ορίζουμε τις αρχικές μας μεταβλητές, όπως για παράδειγμα τον χάρτη, το σημείο εκκίνησης του παίχτη και άλλα.

Φάση 3- Λήψη εισόδου από την χρήστη

Είναι το σημείο όπου το παιχνίδι μας λαμβάνει από τον χρήστη τα λεγόμενα **interrupts** ή αλλιώς εισόδους από το πληκτρολόγιο, το ποντίκι ή ακόμα και από ένα χειριστήριο παιχνιδιού. Αυτή η φάση, που είναι και η μοναδική, είναι η μόνη γέφυρα που συνδέει το παιχνίδι με το χρήστη. Με άλλα λόγια ο χρήστης μπορεί με αυτόν τον τρόπο να «επικοινωνήσει» με τον παιχνίδι.

Φάση 4- Επεξεργασία λογικών του παιχνιδιού

Σε αυτήν την φάση ελέγχουμε και ορίζουμε το «τι μέλει γενέσθαι» στο παιχνίδι μας. Δηλαδή το πού ακριβώς μετακινήθηκε ο παίχτης μας τα τελευταία δευτερόλεπτα, πόσες ακριβώς σφαίρες έχουν μείνει, αν το πλοίο που οδηγούμε συγκρούστηκε πάνω σε κάποιο βράχο και άλλα.

Φάση 5- Εμφάνιση γραφικών

Εδώ, «ζωγραφίζουμε» τα μοντέλα μας και τα αντικείμενα μας στην οθόνη. Αυτή η φάση έχει άμεση σχέση με την προηγούμενη και το τι θα ζωγραφιστεί έχει να κάνει με την προηγούμενη φάση.

Φάση 6- Έλεγχος για έξοδο ή για συνέχεια

Σε αυτήν την φάση το παιχνίδι μας ελέγχει το τι έγινε στις προηγούμενες φάσεις, και ανάλογα αποφασίζεται αν θα συνεχιστεί ο βρόγχος ή θα τερματιστεί. Είναι κατανοητό το γεγονός ότι εάν τερματιστεί ο βρόγχος, θα γίνει έξοδος από το παιχνίδι, ενώ διαφορετικά, ο βρόγχος θα συνεχιστεί κανονικά από την φάση 3 και έπειτα.

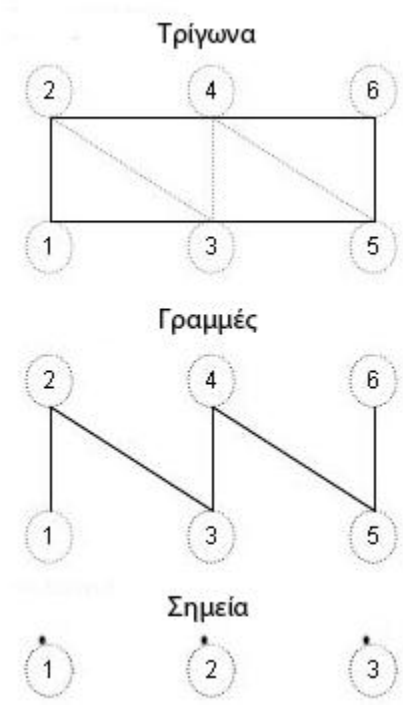
Φάση 7- Καθαρισμός γραφικών, απελευθέρωση μνήμης

Αυτή η φάση είναι συμπληρωματική και δεν καλείται συνεχώς στην **Main Game Loop** παρά μόνο όταν αυτό ζητηθεί. Από τον τίτλο καταλαβαίνουμε ότι εδώ μπορούμε να καθαρίσουμε μοντέλα ή και μνήμη η οποία δεν χρησιμοποιείται πλέον και να την αφήσουμε για μετέπειτα χρήση ώστε να μην έχουμε υπερχείλιση μνήμης.

Αυτό λοιπόν ήταν ένα βασικό κομμάτι για να κατανοήσουμε το πώς λειτουργεί ένα παιχνίδι, με ποια, δηλαδή, λογική. Με τι εντολές εμφανίζουμε αντικείμενα, ελέγχουμε για συγκρούσεις και αποδεσμεύουμε μνήμη έχει να κάνει με την εκάστοτε μηχανή γραφικών και δεν είναι κάτι γενικό. Η λογική μόνο παραμένει η ίδια.

Παρακάτω θα μιλήσουμε για **τα βασικά συστατικά** ή **βασικά στοιχεία** από τα οποία απαρτίζονται όλα τα αντικείμενα και τα μοντέλα που βλέπουμε εμείς στην οθόνη μας. Κάθε φορά που μιλάμε για **βασικά στοιχεία** για τα παιχνίδια είναι σαν να μιλάμε για μόρια και άτομα στην επιστήμη

της φυσικής. Είναι δηλαδή κάτι από το οποίο απαρτίζονται όλα όσα βλέπουμε ως γραφικά στην οθόνη μας και είναι ίδια για τα πιο απλά μοντέλα αλλά και τα πιο σύνθετα. Το πιο γνωστό **βασικό στοιχείο** που όλες οι μηχανές γραφικών υποστηρίζουν είναι τα **τρίγωνα**. Πολύπλοκα μοντέλα, όπως η βροχή που βλέπουμε να πέφτει στο παιχνίδι μας, το αεροπλάνο που πετάει στους ουρανούς του παιχνιδιού μας, είναι αποτέλεσμα εκατομμυρίων τριγώνων τα οποία ενωμένα μεταξύ τους μας δίνουν αυτήν την «ψευδαίσθηση» του βουνού, του νερού, του ουρανού. Ακόμη και τα προγράμματα κατασκευής μοντέλων δουλεύουν με τρίγωνα και σε προηγούμενο κεφάλαιο που μιλούσαμε για εξαγωγή μοντέλων σε μορφή κατανοητή από την μηχανή γραφικών από τα προγράμματα κατασκευής μοντέλων, αυτά δεν εξάγουν τίποτε άλλο παρά ένα κείμενο με τον αριθμό των τριγώνων, το χρώμα τους, το πού βρίσκονται οι κορυφές τους και άλλα στοιχεία. Συχνά, επίσης, βλέπουμε κάρτες γραφικών να γράφουν στα χαρακτηριστικά τους ότι μπορούν να εμφανίσουν «χ εκατομμύρια τρίγωνα το δευτερόλεπτο». Η χρήση τριγώνων, λοιπόν, απ' ό, τι διαπιστώνουμε είναι πλέον δεδομένη στη βιομηχανία παιχνιδιών. Βεβαίως υπάρχουν και άλλα **βασικά στοιχεία** με τα οποία μπορούμε να σχεδιάσουμε γραφικά, όπως για παράδειγμα τελείες (σημεία) και γραμμές. Μα για ποιο λόγο, όμως, χρησιμοποιούνται ευρέως τα τρίγωνα και όχι, για παράδειγμα, οι γραμμές; Η απάντηση είναι πολύ απλή. Τα τρίγωνα και λόγω των τριών κορυφών τους μας είναι χρήσιμα για να κατασκευάσουμε άλλα βασικά σχήματα. Σκεφτείτε μόνο ότι δύο τρίγωνα είναι ένα τετράγωνο. Ένας κύλινδρος τρισδιάστατος θα ήταν πολλά, ή μάλλον εκατοντάδες, τρίγωνα. Λόγω του γεγονότος ότι πλέον ακόμα και για να σχεδιάσουμε μια «τρισδιάστατη γραμμή» οι γραμμές δεν είναι αρκετές (αυτό θα ήταν συνετό μόνο στα πρώτα παιχνίδια της δεκαετίας του '70) και, όπως είπαμε και πριν, λόγω των κορυφών των τριγώνων και ότι αυτές μπορούν να ενωθούν ώστε να σχεδιάσουν πολύπλοκα μοντέλα, μπορούμε να γλιτώσουμε μεγάλο **bandwidth** στην κάρτα γραφικών μας. Τα νούμερα τα οποία εξοικονομούμε, για παράδειγμα στο σχεδιασμό ενός απλού τετραγώνου, είναι μικρά από την χρήση τριγώνων αντί γραμμών ή σημείων. Εάν σκεφτούμε, όμως, ότι τα σύγχρονα παιχνίδια αποτελούνται από εκατοντάδες μοντέλα τα οποία κάθε δευτερόλεπτο κάνουν πολλές κινήσεις, τα νούμερα εξοικονόμησης είναι τρομακτικά και δεν θα ήταν δυνατή η εμφάνιση τους, παρά μόνο με την βοήθεια πανάκριβων καρτών γραφικών οι οποίες, φυσικά, θα ανέβαζαν κατά πολύ το κόστος του υπολογιστή μας. Παρακάτω παρατίθεται ένα διάγραμμα που δείχνει μερικά **βασικά στοιχεία** γραφικών.



Εικόνα 5: Τα βασικά στοιχεία των γραφικών

Κλείνοντας το κεφάλαιο αυτό, συμπερασματικά θα λέγαμε ότι τα σύγχρονα παιχνίδια, από τα πιο απλά έως τα πιο σύνθετα και εντυπωσιακά, χρησιμοποιούν στην ουσία την τεχνική του ατέρμονου

βρόγχου, για να λαμβάνουν **μηνύματα** από τους χρήστες όπως είδαμε και πριν. Τέλος, για να μπορέσουν να ζωγραφίσουν κάτι στην οθόνη χρησιμοποιούν τα **βασικά στοιχεία** της μηχανής γραφικών. Βεβαίως, θα αναρωτηθεί κάποιος, εάν μόνο αυτά τα δύο είναι επαρκή για να καταφέρουμε αυτά τα εφέ ή τις αντανακλάσεις και τους φωτισμούς που βλέπουμε στα σύγχρονα παιχνίδια. Η απάντηση φυσικά είναι όχι και όπως θα δούμε, υπάρχουν κάποιες άλλες τεχνικές που χρησιμοποιούνται, απλώς αυτά που παρουσιάστηκαν σε αυτό το κεφάλαιο είναι τα βασικά και σε αυτά βασίζονται και όλα τα υπόλοιπα.

Κεφάλαιο IV

Παρουσίαση της μηχανής γραφικών Esenthel

4.1 Λίγα λόγια για τη μηχανή Esenthel

Σε αυτό το κεφάλαιο θα δούμε αναλυτικά τη μηχανή γραφικών Esenthel και τις δυνατότητες που προσφέρει στο χρήστη. Η μηχανή Esenthel είναι μια ολοκληρωμένη πλατφόρμα ανάπτυξης παιχνιδιών δίνοντας τη δυνατότητα δημιουργίας τίτλων κατηγορίας ΑΑΑ (υψηλής ποιότητας και με μεγάλο προϋπολογισμό). Έχει σχεδιαστεί ειδικά για επαγγελματική ανάπτυξη παιχνιδιών, η οποία επιτυγχάνεται δίνοντας στους προγραμματιστές απόλυτο έλεγχο στη μηχανική του κώδικα του παιχνιδιού, γραφικά επόμενης γενιάς και μια πλούσια εργαλειοθήκη η οποία μειώνει δραστικά τη διαδικασία ανάπτυξης του παιχνιδιού. Παρόλο όμως που στοχεύει κατά κύριο λόγο την επαγγελματική αγορά, είναι αρκετά εύκολο να χρησιμοποιηθεί από ανεξάρτητες ομάδες ή αυτόνομους χρήστες χωρίς προηγούμενη εμπειρία στην ανάπτυξη παιχνιδιών.

Το Esenthel δίνει τη δυνατότητα στο χρήστη να δημιουργήσει το παιχνίδι του μία φορά αλλά να μπορεί να το εκδώσει για πολλές διαφορετικές πλατφόρμες (Windows, Mac, Android κλπ). Διαθέτει ένα από τα πιο εξελιγμένα συστήματα παρουσίασης γραφικών παρέχοντας τη καλύτερη ποιότητα και απόδοση αν και οι απαιτήσεις σε επεξεργαστική ισχύ ίσως είναι μεγαλύτερες από συνήθως. Λόγω της συνεργασίας με το σύστημα PhysX της εταιρίας Nvidia διαθέτει εξελιγμένη προσομοίωση όσο ν αφορά τη φυσική των αντικειμένων και δίνει τη δυνατότητα ύπαρξης τεράστιου αριθμού δυναμικών αντικειμένων στη σκηνή σε πραγματικό χρόνο. Η εργαλειοθήκη του Esenthel διαθέτει μεγάλη γκάμα εργαλείων εύκολα στη χρήση τα οποία επιτρέπουν στο χρήστη να επεξεργάζεται αυτόματα και να εισάγει στοιχεία στο παιχνίδι του δημιουργώντας ατελείωτους κόσμους και να προσαρμόζει απόλυτα τα αντικείμενα όπως αυτός επιθυμεί.

Εύκολα μπορεί να ισχυριστεί κανείς ότι τη μηχανή Esenthel τη χαρακτηρίζει απόλυτα ο όρος σταθερότητα καθώς τα πάντα δουλεύουν όπως πρέπει, είναι δοκιμασμένα από τους προγραμματιστές που τη δημιούργησαν πάμπολλες φορές αλλά ακόμη και αν βρεθεί κάποιο σφάλμα είναι έτοιμοι να το επιδιορθώσουν άμεσα χωρίς καθυστέρηση. Πρέπει να αναφέρουμε σ αυτό το σημείο ότι υπάρχουν αφοσιωμένα άτομα έτοιμα να επιλύσουν οποιοδήποτε πρόβλημα προκύψει. Τέλος, πρέπει να αναφερθεί ότι η μηχανή συνεχώς εξελίσσεται και κάθε μερικές μέρες κάτι καινούριο προστίθεται ή εξελίσσεται. Σχετικά με την χρήση της, αρχικά φαίνεται εύκολη στη χρήση αλλά και όταν αρχίσουν και δυσκολεύουν τα πράγματα υπάρχουν περισσότερα από 100 επεξηγηματικά αρχεία (tutorials) άριστα φτιαγμένα και απόλυτα κατανοητά.

4.2 Χαρακτηριστικά της μηχανής

4.2.1 Rendering

Αρχικά πρέπει να μιλήσουμε για το ανεπτυγμένο σύστημα render που διαθέτει. Στην ουσία διαθέτει τρεις διαφορετικούς renderers για να μπορεί να προσφέρει υψηλότερο frame – rate σε μηχανήματα με όχι και τόσο μεγάλη επεξεργαστική ισχύ.

- Ο πρώτος είναι ο και ο πιο απλός αλλά και ο πιο γρήγορος. Είναι περιορισμένος στην ύπαρξη μόνο μιας πηγής φωτός χωρίς να υπάρχουν σκιές, κατοπτρικοί φωτισμοί ή ανάγλυφες υφές. Είναι κατάλληλος για υπολογιστές με χαμηλές δυνατότητες.

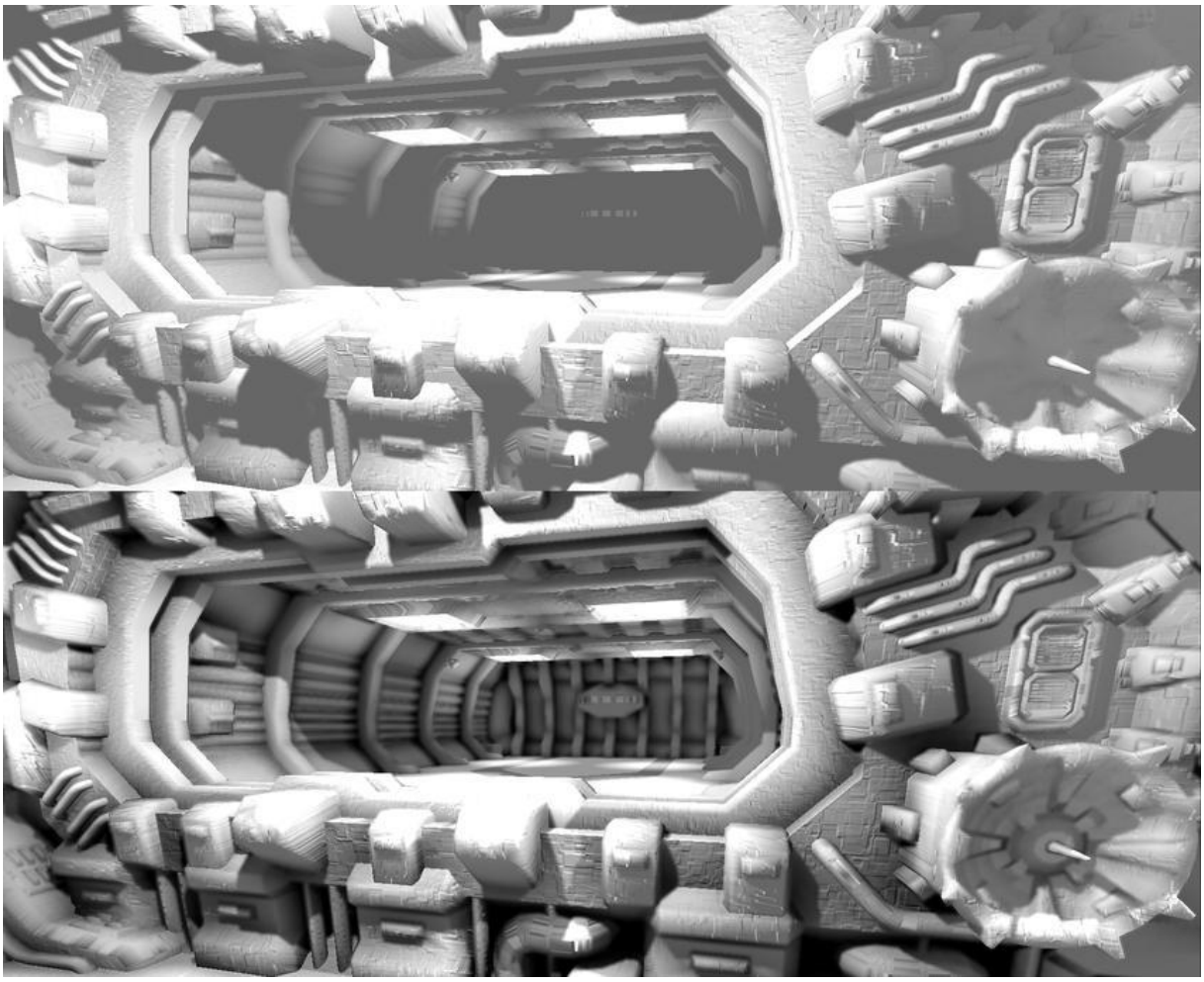
- Ο δεύτερος σε σειρά υποστηρίζει απεριόριστο φωτισμό, υφές επίπεδες και ανάγλυφες αναλόγως το μέγεθος της επιφάνειας. Είναι κατάλληλος για απλές σκηνές με απλούς φωτισμούς.
- Ο τρίτος αποτελεί και τον καλύτερο renderer υποστηρίζοντας όλων των ειδών τα φώτα, τις υφές και όλα τα ειδικά εφέ. Είναι κατάλληλος για πολύ επίπεδες και περίπλοκες σκηνές με μεγάλο αριθμό από φώτα.

Υποστηρίζονται τα εξής είδη φωτισμών:

i. Κατευθυντήριο (Directional)
ii. Σημείου (Point)
iii. Σημείου με γρήγορο σβήσιμο (Point with fast fading)
iv. Κωνικό (Cone)

Οι υπολογισμοί για το φωτισμό γίνονται για κάθε pixel έτσι ώστε να υπάρχει μεγαλύτερη ποιότητα στην εικόνα. Όλων των ειδών τα φώτα μπορούν να δημιουργήσουν σκιά και δίνεται η δυνατότητα επιλογής ανάμεσα σε κλιμακωτές σκιές, σκιές με τρεμούλιασμα και άλλα.

Χρησιμοποιώντας την τεχνική SSAO (Screen Space Ambient Occlusion) αυξάνει δραματικά τον ρεαλισμό τον γραφικών κάνοντας πιο σκοτεινές τις αποφραγμένες περιοχές.



Εικόνα 6: Τεχνική SSAO

Οι κατασκευαστές έχουν δώσει μεγάλη σημασία στο βάθος του πεδίου (Depth of field) το οποίο είναι άλλο ένα χαρακτηριστικό που βοηθάει στο ρεαλισμό που προσφέρει η μηχανή. Υποστηρίζεται μια τεράστια γκάμα σχετικά με τις υφές (έγχρωμες, κανονικές, ανάγλυφες, γυαλιστερές, με αντανάκλαση κ.α) και τον τύπο τους (bmp, png, jpg, psd, tga…) καθώς και επιπλέον σωματίδια (particles). Σε όλα όσα αναφέρονται παραπάνω δίνεται η δυνατότητα στο χρήστη να τα δημιουργήσει μόνος του ή να επιλέξει από τα ήδη υπάρχοντα.

4.2.2 Φυσική

Έχει δοθεί ιδιαίτερη προσοχή από τους κατασκευαστές στη φυσική των αντικειμένων που βρίσκονται στο χώρο. Πρόσφατα ο προσομοιωτής του φυσικού στοιχείου στη μηχανή αναβαθμίστηκε στην έκδοση 2.8.3 PhysX SDK το οποίο δίνει μεγάλα πλεονεκτήματα στους χρήστες και επιτρέπει τη δημιουργία παιχνιδιών για 64bit επεξεργαστές καθώς και ανεβάζει σε άλλο επίπεδο την κίνηση στα ρούχα και σε άλλα σημεία που είχαν αγνοηθεί στο παρελθόν. Αξίζει να σημειωθεί η δουλειά που έχει γίνει στην κίνηση των αρθρώσεων στους χαρακτήρες καθώς και στην καταστροφή αντικειμένων. Υπάρχει επίπεδο στη λεπτομέρεια του κόσμου και των χαρακτήρων, για παράδειγμα ένας σκελετός μακριά από την κάμερα έχει λιγότερα κόκκαλα με σκοπό την εξοικονόμηση μνήμης. Υποστηρίζει την εισαγωγή animation από προγράμματα όπως Autodesk, Unreal Engine, BioVision, DirectX και αρκετά άλλα ώστε να καλύπτει όλες τις απαιτήσεις.

4.2.3 Επιπλέον χαρακτηριστικά

Στο χρήστη δίνεται η δυνατότητα να διαμορφώσει το γραφικό του περιβάλλον σε μεγάλο βαθμό και υποστηρίζονται τροποποιημένα αντικείμενα όπως κουμπιά, μενού, checkbox, λίστες, εικόνες, combobox κλπ.. Υποστηρίζει λειτουργίες με πληκτρολόγιο, ποντίκι ή ακόμη και μοχλό. Υπάρχει δυνατότητα λειτουργίας με οθόνη αφής και κινητές συσκευές αναγνωρίζοντας τις καταστάσεις των πατημάτων (πάτημα κουμπιού – απελευθέρωση, διπλό κλικ).

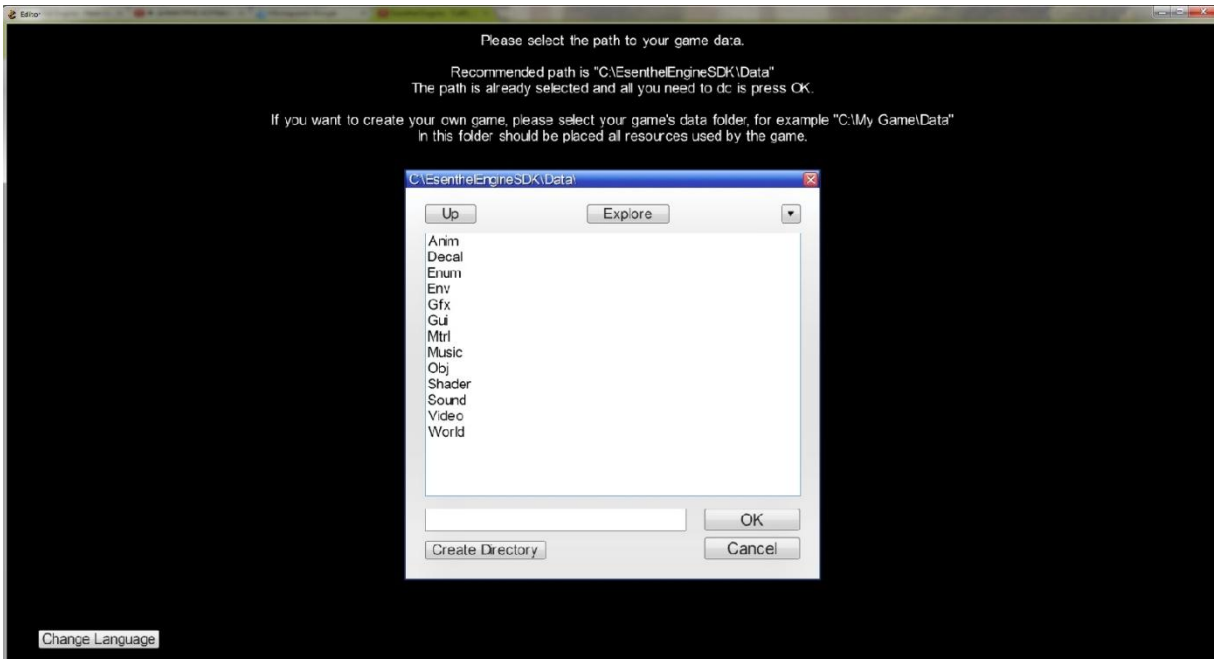
Σχετικά με τα αρχεία του παιχνιδιού προσφέρει κωδικοποίηση χρησιμοποιώντας προσωπικά κλειδιά δίνοντας τη δυνατότητα κανείς να μην έχει πρόσβαση στα δεδομένα ασφαλίζοντας ακόμη και ξεχωριστά τις κλάσεις επιτρέποντας τη χρησιμοποίηση αλγορίθμων κωδικοποίησης. Όλα τα αρχεία μπορούν να αρχειοθετηθούν αυτόματα σε ένα ‘‘πακεταρισμένο’’ αρχείο (.pak file). Εμπεριέχει μια αρκετά σύνθετη και δυνατή αριθμομηχανή η οποία πέρα από απλές πράξεις και μετατροπές μπορεί να υπολογίσει μαθηματικές παραστάσεις όπως: "sin(2*x) + dot(vec(1,2,3),vec(4,5,6)) + 15*(1+2*pow(x,y))". Μπορούμε να χρησιμοποιήσουμε όλων των τύπων τους αριθμούς int, float, double καθώς και πολυδιάστατους πίνακες, διανύσματα και όλες τις απαραίτητες μαθηματικές συναρτήσεις. Το σύστημα διαχείρισης της μνήμης είναι αρκετά περίπλοκο καθώς δεσμεύει και απελευθερώνει αυτόματα τη μνήμη προσφέροντας πολλαπλές λειτουργίες και συναρτήσεις για τα στοιχεία.

Τέλος, το Esenthel έχει ασχοληθεί αρκετά με την επεξεργασία των πολυγώνων, την εισαγωγή από άλλα προγράμματα, έχει δημιουργήσει μια σταθερή γεωμετρία, δίνει τη δυνατότητα εισαγωγής και επεξεργασίας βίντεο, μουσικής και κειμένου. Η σύνδεση με FTP servers είναι δυνατή και χρησιμοποιεί είτε TCP/IP είτε UDP sockets.

4.3 Δημιουργία νέου παιχνιδιού

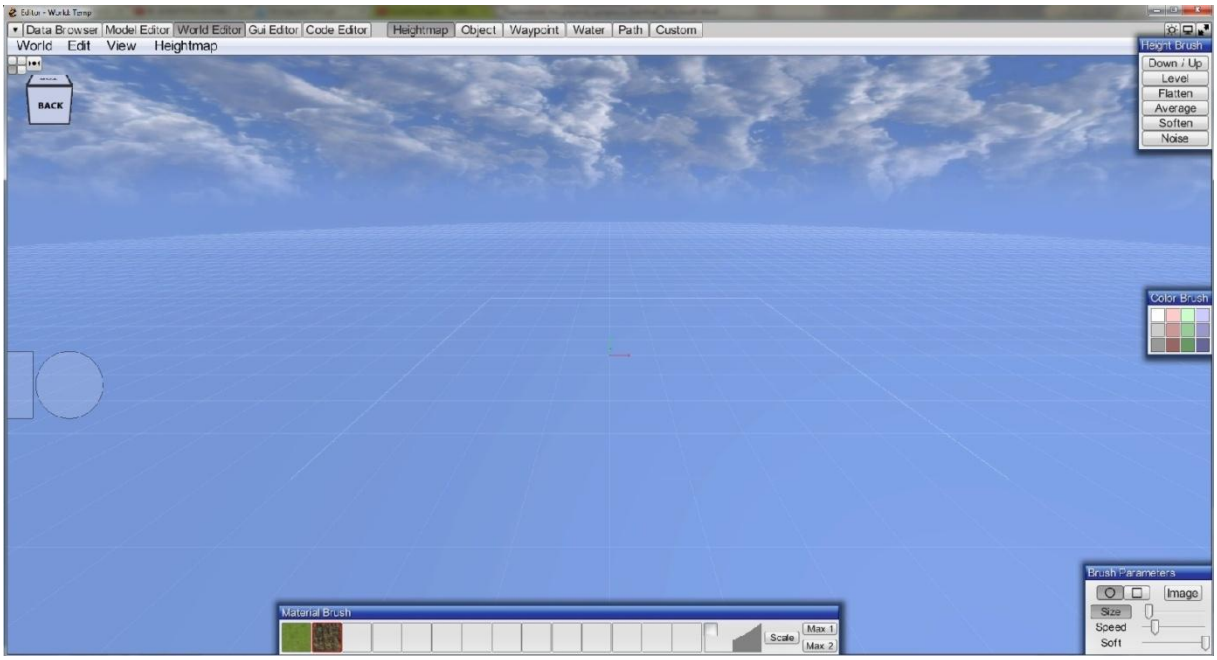
Εδώ θα δούμε αναλυτικά την διαδικασία δημιουργίας νέου παιχνιδιού στον Editor καθώς και τον τρόπο εισαγωγής υψομετρικών εικόνων (heightmaps) ακριβώς στις θέσεις που επιθυμούμε αν και δεν θα χρειαστεί να εμβαθύνουμε περισσότερο σ αυτό το κομμάτι καθώς τα υπόλοιπα θα τα γίνουν παρακάτω μέσω του Visual Studio.

Παρακάτω φαίνεται το πρώτο παράθυρο που μας εμφανίζεται μετά την εγκατάσταση όταν ανοίξουμε τον editor πρώτη φορά.



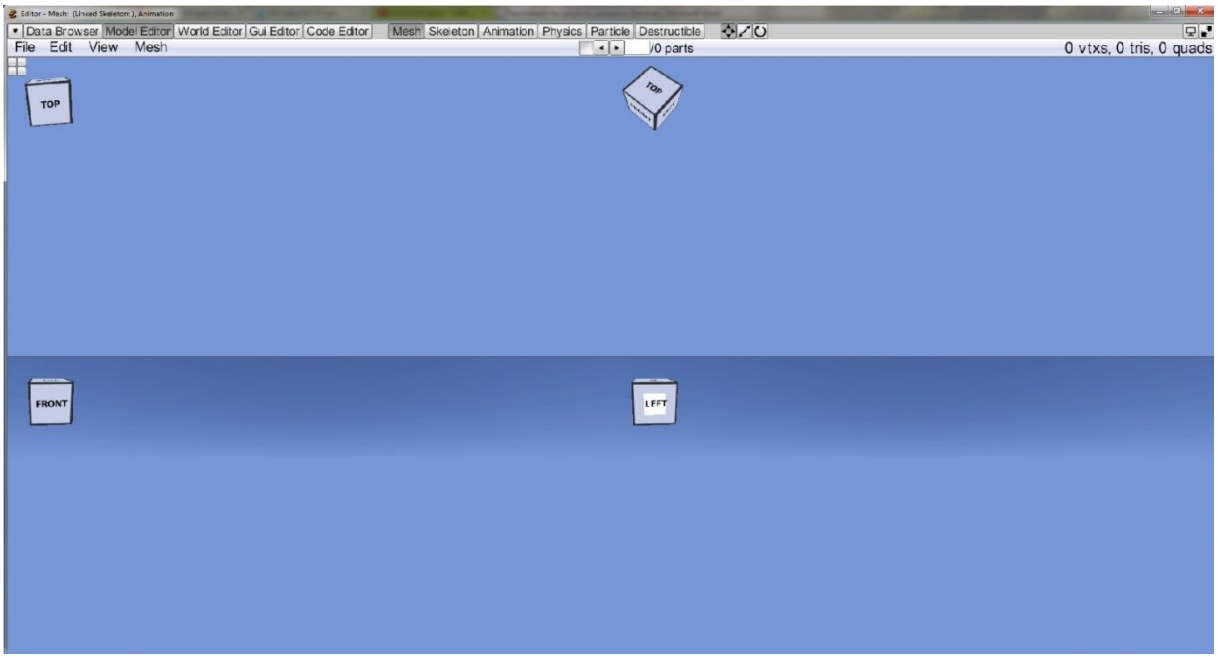
Εικόνα 7: Ορισμός του DataPath

Μας ζητάει να του ορίσουμε την διαδρομή αρχείων (DataPath) απ' όπου θα αντλεί τα δεδομένα που χρειάζεται καθώς και να επιλέξουμε τη γλώσσα που επιθυμούμε. Μετά την εγκατάσταση και στην συγκεκριμένη διαδρομή που φαίνεται στην εικόνα υπάρχουν ήδη υλικά που μπορούμε να χρησιμοποιήσουμε που έρχονται με την εγκατάσταση. Μόλις του καθορίσουμε την διαδρομή μπαίνουμε στο κεντρικό παράθυρο του editor.



Εικόνα 8: Esenthel World Editor

Αυτή είναι η πρώτη ματιά στον Editor της μηχανής. Για κάποιον ο οποίος έχει ξανά ασχοληθεί με προγράμματα δημιουργίας 3D γραφικών ίσως του φανεί αρκετά οικείο το περιβάλλον. Πιο οικείο από πολλά άλλα προγράμματα θα έλεγα. Υπάρχει κάτω μία παλέτα με τα υλικά που θέλουμε να χρησιμοποιήσουμε, δεξιά βλέπουμε μπάρες με τις οποίες ρυθμίζουμε την ταχύτητα προσθήκης των υλικών, το μέγεθος της βούρτσας καθώς και τη σκληρότητα. Πιο πάνω βλέπουμε μια παλέτα χρωμάτων. Αριστερά πάνω υπάρχει ένας κύβος τον οποίο τραβώντας τον προς την κατάλληλη κατεύθυνση κινούμαστε μέσα στο χώρο. Από κάτω βλέπουμε ένα κύκλο και δίπλα ένα παραλληλόγραμμο το οποίο χρησιμεύει στην μετακίνηση όλου του block που περιλαμβάνει η κάμερα. Στη μπάρα πάνω παρατηρούμε τις καρτέλες World, Edit, View, Heightmap οι οποίες περιέχουν διαφόρων ειδών λειτουργίες. Τέλος ακόμη πιο πάνω βλέπουμε τους διαφορετικούς Editors που διαθέτει η μηχανή. Στην εικόνα κάτω φαίνεται ο Model Editor με ένα αρκετά οικείο παρουσιαστικό.



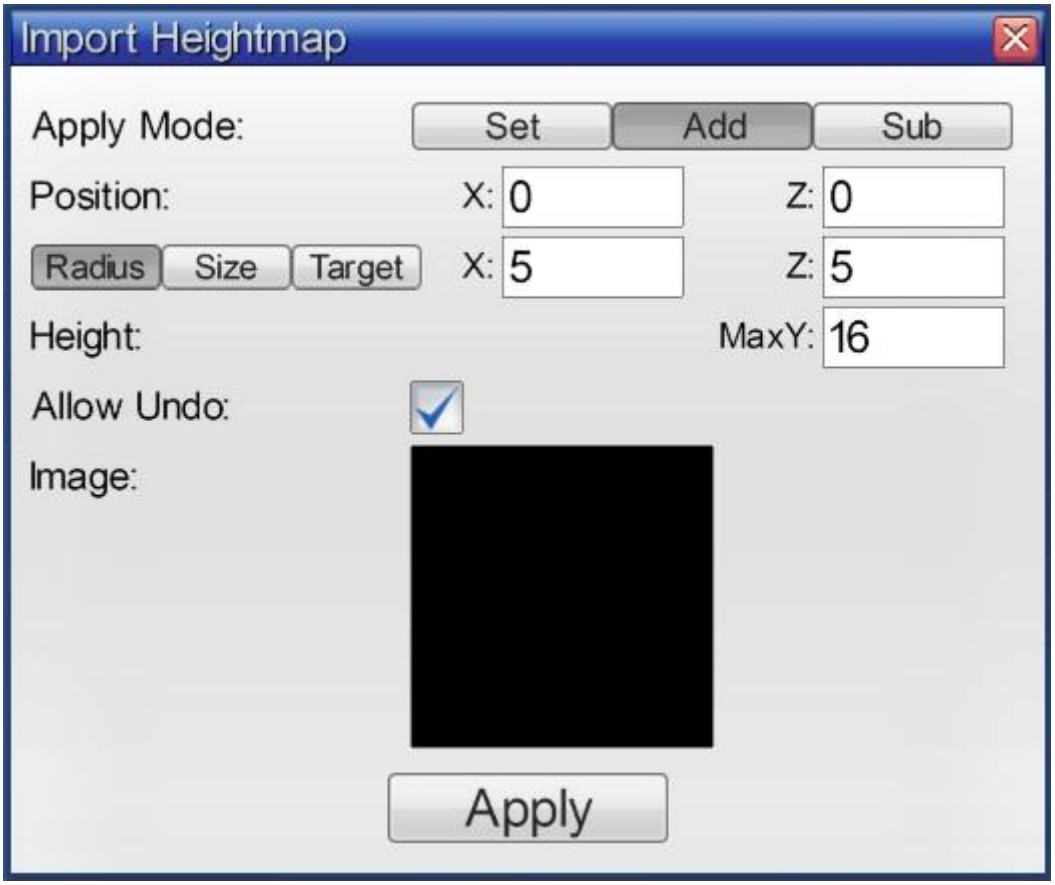
Εικόνα 9: Esenthel Model Editor

Παρόλο που το σχεδιαστικό κομμάτι της μηχανής δεν θα μας απασχολήσει ιδιαίτερα στη συγκεκριμένη πτυχιακή θα δούμε με ποιο τρόπο εισάγουμε υψομετρικές εικόνες όπως η παρακάτω στον editor.

Εικόνα 10: Heightmap

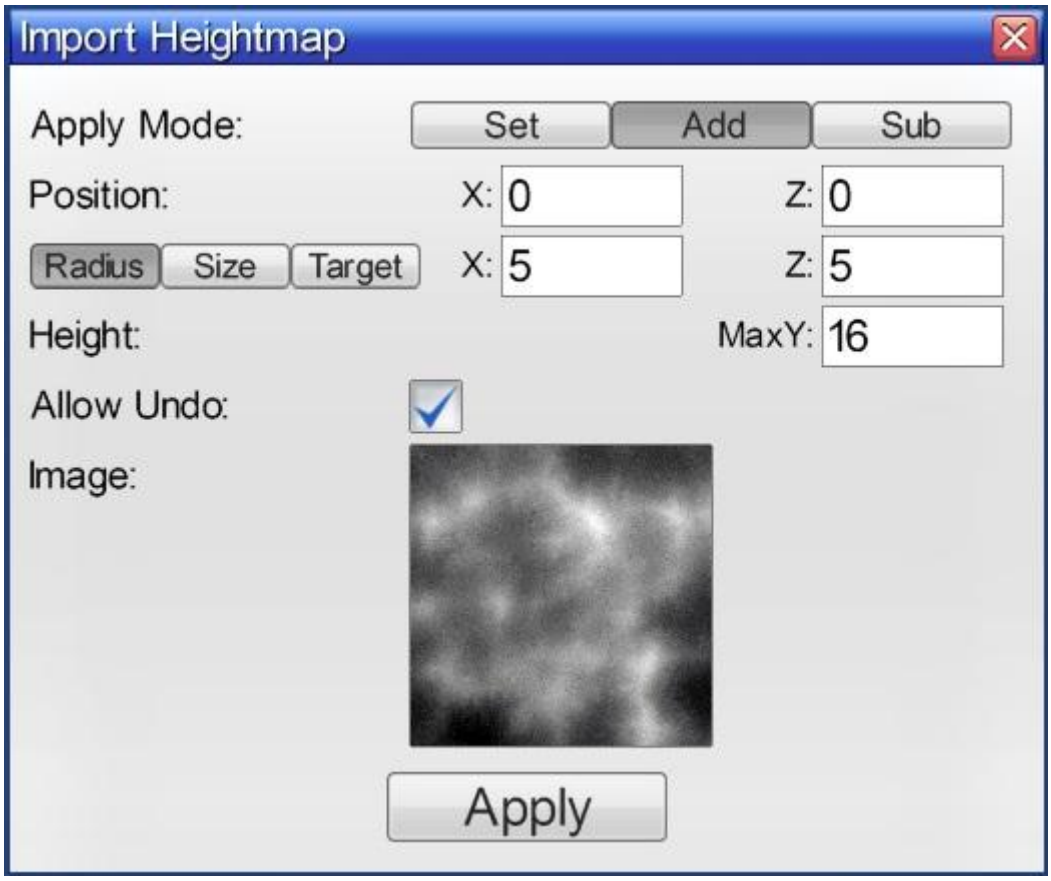
Αξίζει σ' αυτό το σημείο να αναφέρουμε ότι ως υψομετρική εικόνα ή όπως είναι πιο γνωστή heightmap ονομάζουμε μια ασπρόμαυρη (grayscale) εικόνα στην οποία το μαύρο αντιπροσωπεύει το μικρότερο ύψος και το άσπρο το μεγαλύτερο. Αυτό σημαίνει ότι κάθε pixel της εικόνας παίρνει τιμές από 0 μέχρι 255 όσο είναι και το εύρος των γκρι αποχρώσεων (grayscale) και ερμηνεύεται ανάλογα. Στη συγκεκριμένη εργασία ασχολούμαστε πολύ με την επεξεργασία αυτών των εικόνων χρησιμοποιώντας Java όπως θα δούμε και παρακάτω.

Γυρνώντας τώρα πάλι πίσω στον Editor της μηχανής και επιλέγοντας την επιλογή "Import Height" από την καρτέλα "Heightmaps" μας εμφανίζει ένα μικρότερο παράθυρο με διάφορες επιλογές και μία μαύρη εικόνα στη μέση.



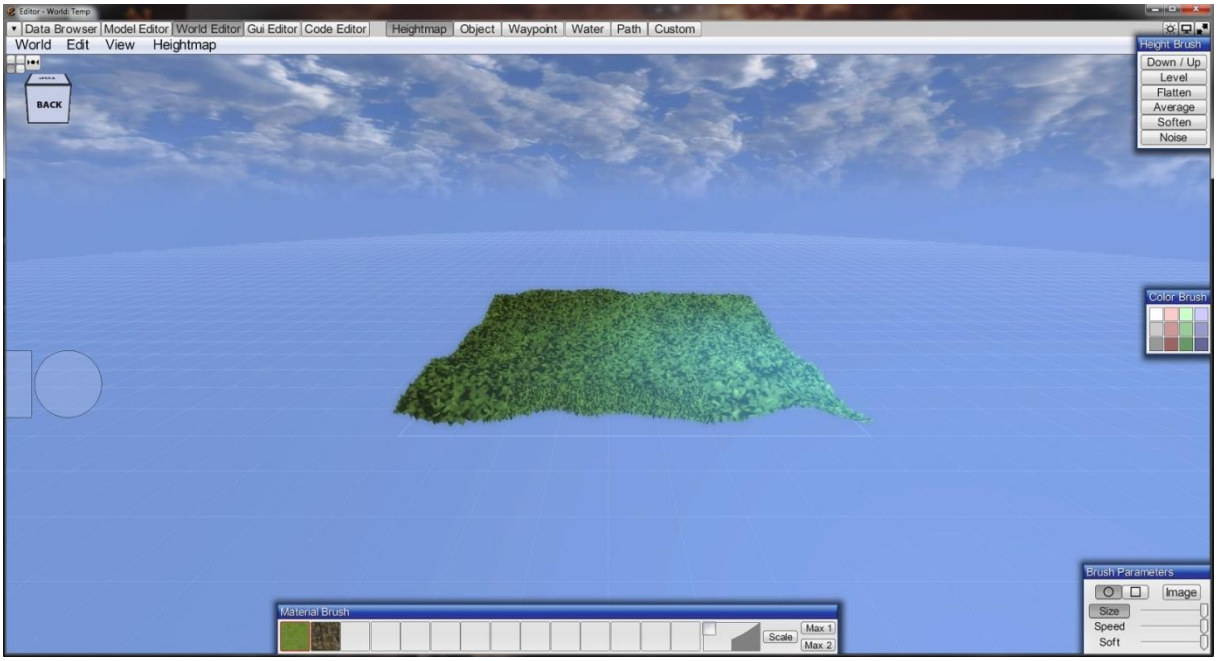
Εικόνα 11: Import Heightmap

Πατώντας πάνω στη μαύρη εικόνα και εντοπίζοντας στα αρχεία μας την εικόνα που θέλουμε να χρησιμοποιήσουμε, την επιλέγουμε και εμφανίζεται το ακόλουθο:



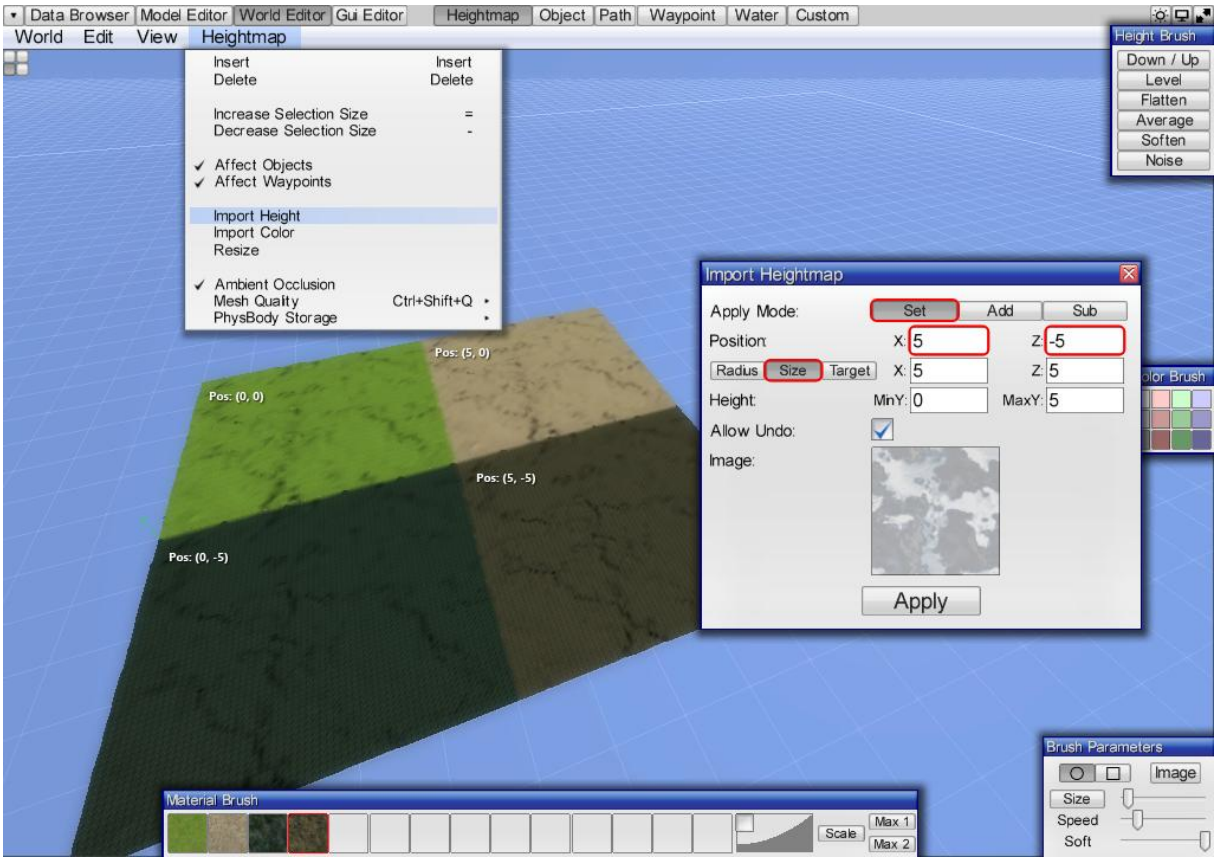
Εικόνα 12: Import Heightmap

Έπειτα πατώντας Apply και έχοντας στη παλέτα των υλικών επιλεγμένο για παράδειγμα το γρασίδι βλέπουμε ότι το τοποθετεί σε όλο το block που ήταν επιλεγμένο.



Εικόνα 13: Heightmap όπως έχει τοποθετηθεί αυτόματα στον Editor

Στη συγκεκριμένη εργασία θα χρειαστεί να εισάγουμε πολλά heightmaps έτσι ώστε να δημιουργήσουμε ένα πεδίο μεγάλης έκτασης που επιθυμούμε. Ο Editor του Esenthel μας δίνει τη δυνατότητα να το κάνουμε. Στο παράθυρο που μας ανοίγει όταν πατάμε το "Import Height" βλέπουμε ότι υπάρχουν ρυθμίσεις που έχουν να κάνουν με συντεταγμένες. Ύψος, πλάτος, ύψος κλπ.. Βάζοντας τις κατάλληλες τιμές στις συντεταγμένες αυτές μπορούμε να έχουμε το επιθυμητό αποτέλεσμα.



Εικόνα 14: Multiple Heightmaps

Θα μπορούσαμε να εξηγούμε για πολύ ώρα τα εργαλεία και τις δυνατότητες που προσφέρει ο editor στους χρήστες αλλά τα tutorials που εμπεριέχονται καθώς και τα βοηθητικά κείμενα (documentation) που εμπεριέχονται είναι απόλυτα κατατοπιστικά οπότε δεν θα διαθέσουμε περισσότερο χρόνο σ αυτό το κομμάτι.

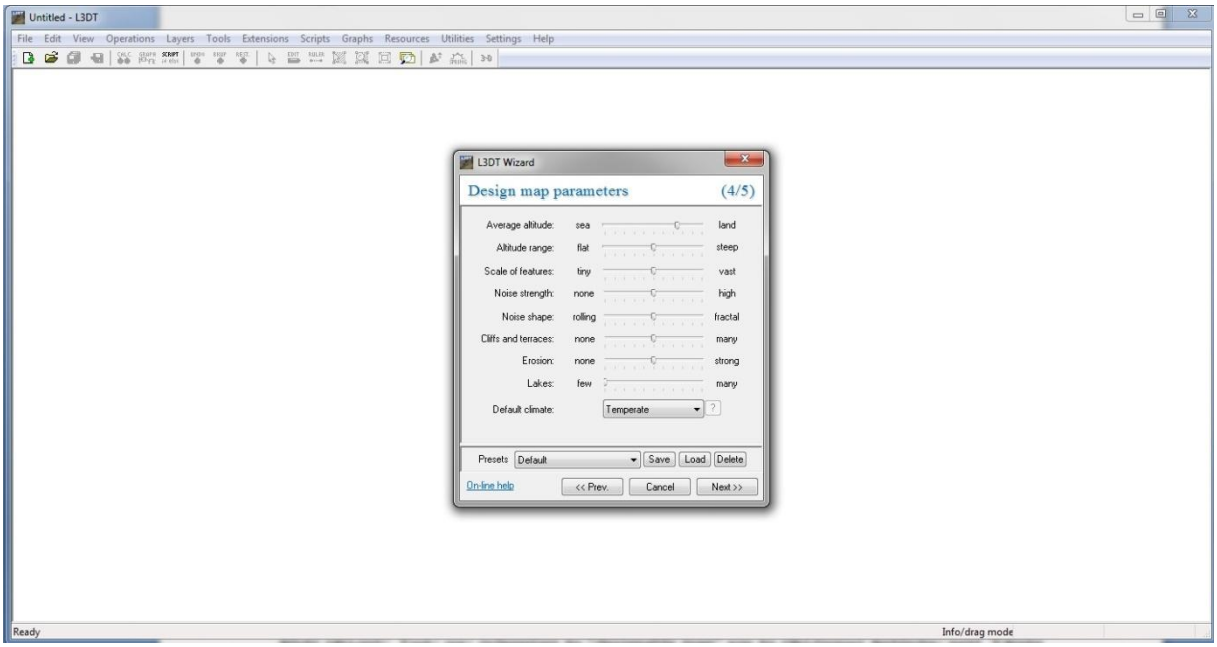
Κεφάλαιο V

Επεξεργασία των χαρτών (Heightmaps)

5.1 Terrain Generator (Γεννήτρια Πεδίων)

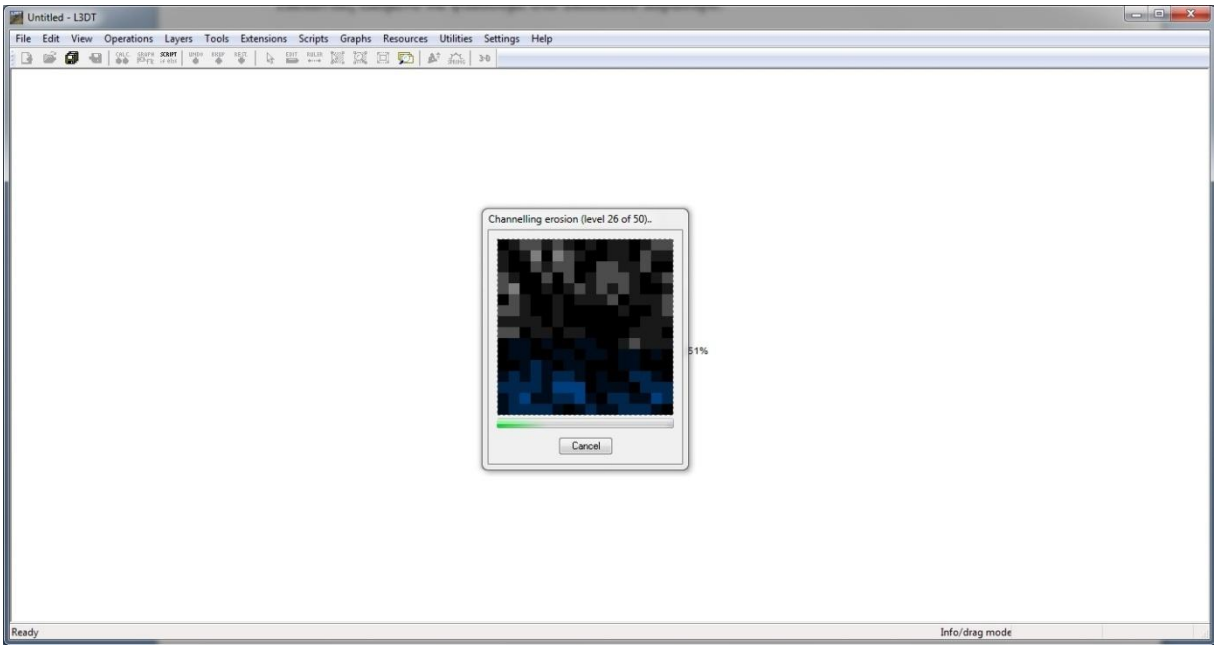
Σε αυτό το κεφάλαιο θα δούμε τον τρόπο με τον οποίο θα δημιουργήσουμε τους απαραίτητους χάρτες που θα χρησιμοποιηθούν και τη διαδικασία που θα ακολουθήσουμε για να τους επεξεργαστούμε. Από τη στιγμή που η εργασία αυτή καταπιάνεται με πεδία μεγάλης έκτασης που είναι πολύ δύσκολο καθώς και χρονοβόρο να ασχοληθεί κανείς με τη δημιουργία του πεδίου (terrain) με το χέρι, υπάρχουν εναλλακτικές λύσεις. Αρκετά διαδεδομένα είναι προγράμματα τα οποία δημιουργούν αυτόματα και εντελώς τυχαία heightmaps. Κάτι τέτοιο μας βολεύει απόλυτα από τη στιγμή που θα χρειαστούμε αρκετά. Η "γεννήτρια" που βρήκαμε και χρησιμοποιήσαμε ονομάζεται L3DT ver11.11 από την εταιρία Bundysoft (http://www.bundysoft.com/L3DT/), την δωρεάν έκδοση.

Ξεκινώντας το πρόγραμμα και διαλέγοντας καινούριο project μας εμφανίζει ένα παράθυρο με πέντε επιλογές. Εμάς μας ενδιαφέρει το "designable map" και το επιλέγουμε πατώντας next. Έπειτα θέλει να ρυθμίσουμε μερικές επιπλέον επιλογές η βασικότερη από τις οποίες είναι το μέγεθος. Πατώντας επόμενο θα φτάσουμε στο ακόλουθο παράθυρο:

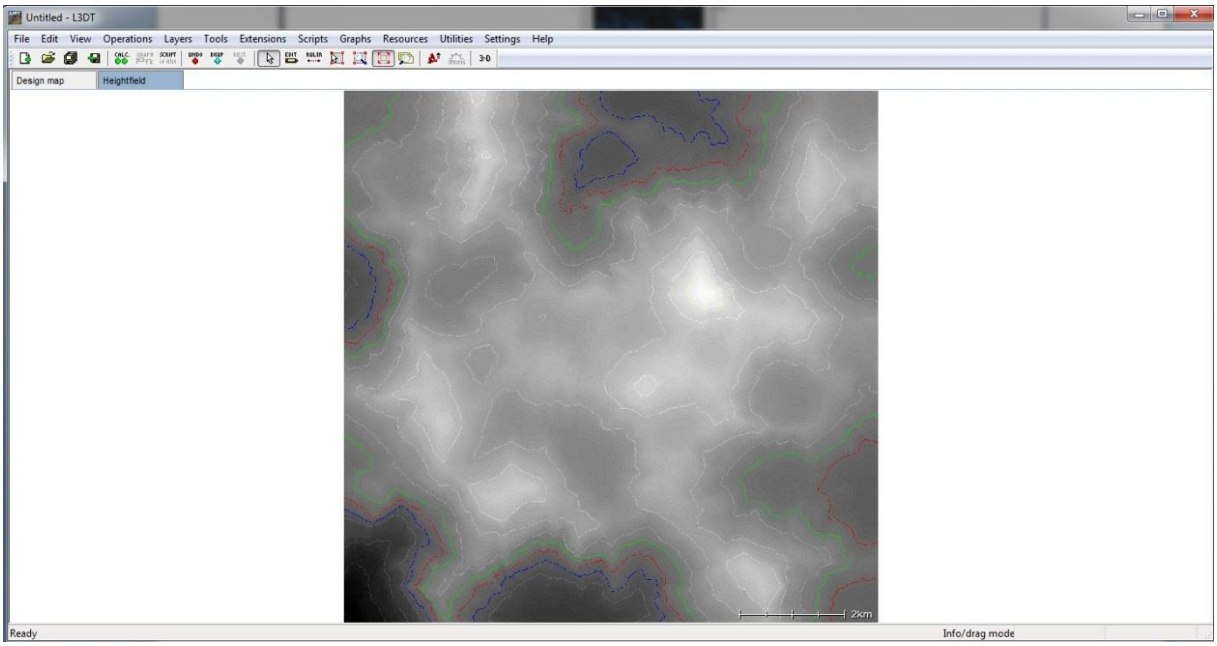


Εικόνα 15: Επιλογές του L3DT

Παραπάνω βλέπουμε τις επιλογές που έχουμε για τη δημιουργία του χάρτη μας. Ρυθμίζουμε όπως εμείς επιθυμούμε το υψόμετρο, την ταχύτητα με την οποία θα αλλάζει το υψόμετρο, το θόρυβο, τις θάλασσες, τα βουνά, τα βράχια και ότι άλλο χρειάζεται. Όταν όλα είναι όπως το επιθυμούμε συνεχίζουμε και του επιλέγουμε να μας φτιάξει πέρα από ένα σχεδιαστικό χάρτη και ένα heightmap και η διαδικασία ξεκινά…



Εικόνα 16: Διαδικασία δημιουργίας του χάρτη



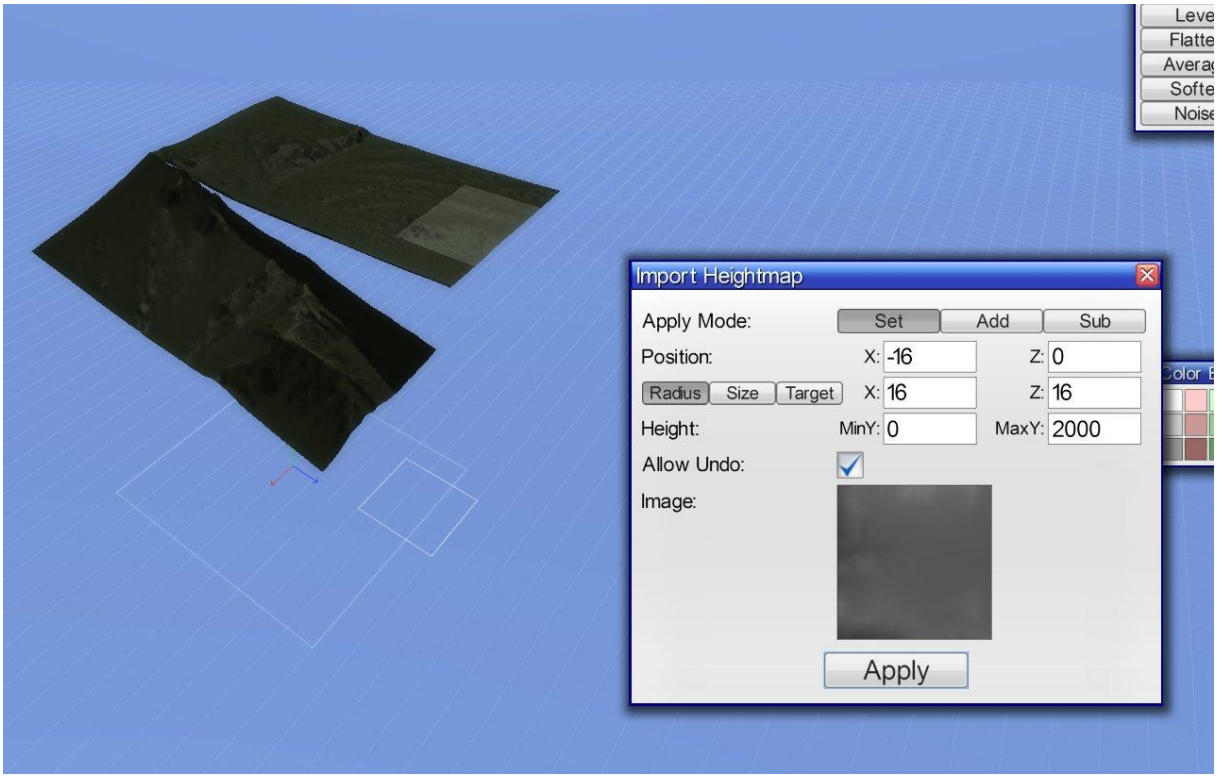
Εικόνα 17: Έτοιμο heightmap από το L3DT

Δεν είναι αυτός ο μοναδικός τρόπος δημιουργίας χαρτών αν και αποτελεί έναν από τους πιο αποτελεσματικούς λόγω της ταχύτητας και της ευκολίας. Ένας αρκετά ενδιαφέρον τρόπος στον οποίο το προσωπικό στοιχείο βέβαια έχει σημαντικό ρόλο είναι η δημιουργία ενός χάρτη μέσω Photoshop. Όπως είναι λογικό χρειάζονται καλές γνώσεις πάνω στο πρόγραμμα και αρκετή εμπειρία. Στη συγκεκριμένη εργασία χρησιμοποιήθηκαν και οι δύο τρόποι.

5.2 Φιλτράρισμα εικόνας στο Περιβάλλον Ανάπτυξης (IDE) NetBeans

5.2.1 Εισαγωγικά

Με τη διαδικασία που ακολουθήσαμε παραπάνω δημιουργούμε εικόνες οι οποίες αποτελούν τη βάση για τον κόσμο που θα δημιουργήσουμε. Στη περίπτωσή μας όμως μία ή δύο εικόνες δεν είναι αρκετές για να τελειώσει ο κόσμος μας. Έτσι μετά τη δημιουργία των εικόνων πρέπει να ακολουθήσει μια διαδικασία επεξεργασίας τους, έτσι ώστε να μην υπάρχουν λάθη ή κενά ανάμεσά τους όπως φαίνεται παρακάτω.



Εικόνα 18: Εισαγωγή heightmap με διαφορετικά ύψη στα σημεία που εφάπτονται

Για να μπορέσουν οι χάρτες που θα μπουν στο πρόγραμμά μας να ταιριάζουν απόλυτα και να μην υπάρχουν κενά πρέπει να επεξεργαστούμε κατάλληλα τα σημεία στα οποία θα εφάπτονται. Στη συγκεκριμένη εργασία θα υλοποιήσουμε αυτή τη διαδικασία προγραμματίζοντας σε Java. Το περιβάλλον ανάπτυξης το οποίο χρησιμοποιούμε είναι το NetBeans (έκδοση 7.2.1). Η διαδικασία που ακολουθούμε είναι να περάσουμε ένα χαμηλοπερατό (low – pass) φίλτρο και στις δυο εικόνες στα κατάλληλα σημεία. Ας δούμε όμως εξ αρχής τα βήματα που ακολουθούμε..

5.2.2 Εισαγωγή εικόνας σε πίνακα και αντίστροφα

Όπως έχει αναφερθεί και πιο πάνω, οι εικόνες με τις οποίες ασχολούμαστε είναι σε αποχρώσεις του γκρι (grayscale). Σε κάθε σημείο της εικόνας, το χρώμα της μας υποδεικνύει και το ύψος που θα έχει όταν σχεδιαστεί τρισδιάστατα. Ως ελάχιστο ύψος παίρνουμε το μαύρο το οποίο αντιστοιχεί στο νούμερο 0 και ως μέγιστο το άσπρο το οποίο αντιστοιχεί στο νούμερο 255. Καταλαβαίνουμε ότι μπορούμε να έχουμε σύνολο 256 διαφορετικά χρώματα στη κλίμακα του γκρι άρα και 256 διαφορετικά ύψη. Ξεκινάμε δημιουργώντας ένα δισδιάστατο πίνακα στον οποίο θα αποθηκεύεται η στο κατάλληλο κελί η τιμή του αντίστοιχου pixel της εικόνας. Για παράδειγμα αν έχουμε ένα heightmap 800x600 pixel θα δημιουργήσουμε ένα πίνακα με 800 γραμμές και 600 στήλες και θα περάσουμε τη τιμή του pixel που βρίσκεται στη κορυφή πάνω αριστερά στη γωνία της εικόνας στη θέση [0][0] του πίνακα που δημιουργήσαμε. Αυτό μπορούμε να το υλοποιήσουμε με τον παρακάτω κώδικα.

```
public int[][] GStoMat() {
    File file = new File(dir, "9.gif"); //εδώ ορίζουμε ποιο αρχείο θέλουμε να επεξεργαστεί
    BufferedImage image;
    try {
       image = ImageIO.read(file);
       Raster image_raster = image.getData();
       this.sampleModel = image_raster.getSampleModel();
       int[][] original;  //που θα τοποθετηθεί η εικόνα

       // παίρνουμε ένα ένα τα pixel
       int[] pixel = new int[3];
       int[] buffer = new int[2];

       // ορίζουμε το μεγεθος του πίνακα
       int w = image_raster.getWidth();
       int h = image_raster.getHeight();
       original = new int[image_raster.getWidth()][image_raster.getHeight()];

       // τοποθετούμε την εικόνα στον πίνακα
```

```
            for (int i = 0; i < image_raster.getWidth(); i++) {
                for (int j = 0; j < image_raster.getHeight(); j++) {
                    pixel = image_raster.getPixel(i, j, pixel);
                    original[i][j] = pixel[0];
                }
            }

            return original;
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        return null;
    }
```

Ονομάσαμε τη συνάρτηση GStoMat δηλαδή GrayScale to Matrix. Στη συνέχεια θα πρέπει να την επεξεργαστούμε αλλά επίσης πρέπει να γίνει και η αντίστροφη διαδικασία από την παραπάνω όταν όλα θα είναι έτοιμα. Στη συνέχεια είναι το κομμάτι κώδικα το οποίο κάνει ένα δισδιάστατο πίνακα ακεραίων με τιμές από 0 έως 255 εικόνα. Το όνομα που δώσαμε στη συνάρτηση είναι το αντίστροφο της προηγούμενης αφού εκτελεί την αντίστροφη λειτουργία.

```
public void matToGS(int pixels[][], SampleModel aSampleModel) {

        this.sampleModel = aSampleModel;
        this.matToGS(pixels);
    }

    public void matToGS(int pixels[][]) {
        int w = pixels.length;
        int h = pixels[0].length;
```

```
    WritableRaster raster = Raster.createWritableRaster(sampleModel, new Point(0, 0));


    for (int i = 0; i < w; i++) {
       for (int j = 0; j < h; j++) {
          raster.setSample(i, j, 0, pixels[i][j]);
       }
    }


    BufferedImage image = new BufferedImage(w, h, BufferedImage.TYPE_BYTE_GRAY);
    image.setData(raster);
    File output = new File(this.dir, "check.png");
    try {
       ImageIO.write(image, "jpg", output);
    } catch (Exception e) {
       e.printStackTrace();
    }
    //return image;
 }
```

5.2.3 Φιλτράρισμα εικόνων – Smoothing, Bluring

Φτάσαμε τώρα στο σημείο οπού πρέπει να επεξεργαστούμε τις εικόνες για να μπορέσουμε να τις χρησιμοποιήσουμε. Η διαδικασία κατά την οποία εφαρμόζουμε ένα low – pass φίλτρο ονομάζεται συνήθως smoothing ή blurring. Στην πιο απλή εφαρμογή του, υπολογίζουμε το μέσο όρο ενός μόνο pixel και των υπολοίπων οκτώ γύρο γύρο. Το αποτέλεσμα αντικαθιστά την αρχική τιμή των pixel και η διαδικασία προχωράει στο επόμενο "εικονοστοιχείο". Το φιλτράρισμα μπορεί να απεικονιστεί με την κατάρτιση ενός "πυρήνα συνέλιξης" (convolution kernel). Ως kernel ορίζουμε ένα πλέγμα το οποίο μας δείχνει πως η φιλτραρισμένη τιμή ενός pixel εξαρτάται από τους γείτονές του. Για να εφαρμόσουμε ένα low-pass φίλτρο μπορούμε να χρησιμοποιήσουμε το παρακάτω πλέγμα:

+1/9	+1/9	+1/9
+1/9	+1/9	+1/9
+1/9	+1/9	+1/9

Όταν εφαρμόζουμε το παραπάνω, κάθε pixel και τα οκτώ γειτονικά πολλαπλασιάζονται με το 1/9 και αθροίζονται. Το μεσαίο pixel αντικαθιστάται από το άθροισμα. Αυτό επαναλαμβάνεται για κάθε pixel της εικόνας που θέλουμε να επεξεργαστεί. Αν δεν θέλουμε το φιλτράρισμα να είναι τόσο ισχυρό θα μπορούσαμε αντίστοιχα να χρησιμοποιήσουμε το εξής πλέγμα:

0	+1/8	0
+1/8	+1/2	+1/8
0	+1/8	0

Το κεντρικό pixel συνεισφέρει το ήμισυ της αξίας του στο αποτέλεσμα και καθένα από τα τέσσερα pixel πάνω, κάτω, αριστερά και δεξιά από το κέντρο συνεισφέρουν το 1/8 το καθένα. Το αποτέλεσμα της δεύτερης περίπτωσης θα είναι πιο μαλακό. Δοκιμάζοντας διαφορετικά φίλτρα μπορούμε να βρούμε το επιθυμητό αποτέλεσμα για κάθε περίπτωση. Θα μπορούσαμε να χρησιμοποιήσουμε πιο μεγάλο πλέγμα, για παράδειγμα 5x5 εξίσου εύκολα ή και ακόμη μεγαλύτερο θα υπήρχε όμως πρόβλημα στον αριθμό των υπολογισμών που θα απαιτούνταν. Σε ένα πλέγμα 3x3 δηλαδή για σύνολο 9 pixel η παρακάτω συνάρτηση είναι αρκετή.

p[i,j]= 0.1*p[i-1,j-1]+0.1*p[i-1,j]+0.1*p[i-1,j+1]+0.1*p[i,j-1]+0.2*p[i,j]+0.1*p[i,j+1]

+ 0.1*p[i+1,j-1]+0.1*p[i+1,j]+0.1*p[i+1,j+1]

Αυτό που μας βόλευε περισσότερο είναι να επεξεργαστούμε τις εικόνες που θα χρησιμοποιήσουμε ανά τετράδες. Στον κώδικα που έχουμε δημιουργήσει και φαίνεται παρακάτω υπάρχει ένα argument στο οποίο αν δεν πάρει καμία τιμή φιλτράρει τις εικόνες στο πλάγιο κομμάτι ενώ δίνοντας την τιμή 1 τις φιλτράρει περιστρέφοντας τις 90 μοίρες και μετά τις επαναφέρει στην αρχική τους θέση έτσι ώστε το bluring να έχει γίνει οριζοντίως.

Ακολουθεί ο κώδικας με τον οποίο υλοποιείται το smoothing:

```
import java.awt.Point;

import java.awt.image.BufferedImage;

import java.awt.image.Raster;

import java.awt.image.SampleModel;

import java.awt.image.WritableRaster;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;
```

```
public class Image {


    private static String dir = "./"; // add here the directory to the folder contains the image

    private SampleModel sampleModel; // stores the sample model of read image. alternative to override with your own


    public int[][] GStoMat(String filename) {

        File file = new File(dir, filename);// file object to get the file, the second argument is the name of the image file

        BufferedImage image;

        try {

            image = ImageIO.read(file);


            Raster image_raster = image.getData();

            this.sampleModel = image_raster.getSampleModel();

            int[][] original; // where we'll put the image


            // get pixel by pixel

            int[] pixel = new int[3];

            int[] buffer = new int[2];


            // declaring the size of arrays

            int w = image_raster.getWidth();

            int h = image_raster.getHeight();

            original = new int[image_raster.getWidth()][image_raster.getHeight()];



            // get the image in the array

            for (int i = 0; i < image_raster.getWidth(); i++) {
```

```
            for (int j = 0; j < image_raster.getHeight(); j++) {
                pixel = image_raster.getPixel(i, j, pixel);
                original[i][j] = pixel[0];
            }
        }

        return original;
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    return null;
}



public void matToGS(int pixels[][], SampleModel aSampleModel, String targetname) {

    this.sampleModel = aSampleModel;
    this.matToGS(pixels, targetname);
}

public void matToGS(int pixels[][], String targetname) {
    int w = pixels.length;
    int h = pixels[0].length;
    WritableRaster raster = Raster.createWritableRaster(sampleModel, new Point(0, 0));

    for (int i = 0; i < w; i++) {
```

```
        for (int j = 0; j < h; j++) {
            raster.setSample(i, j, 0, pixels[i][j]);
        }
    }

    BufferedImage image = new BufferedImage(w, h, BufferedImage.TYPE_BYTE_GRAY);
    image.setData(raster);
    File output = new File(this.dir, targetname);
    try {
        ImageIO.write(image, "gif", output);
    } catch (Exception e) {
        e.printStackTrace();
    }
    //return image;
}
// p[i,j]= 0.1*p[i-1,j-1]+0.1*p[i-1,j]+0.1*p[i-1,j+1]+0.1*p[i,j-1]+0.2*p[i,j-1]+0.1*p[i,j+1]+...

public void ImageBlur(int leftimage[][], int rightimage[][]) {

    if ((leftimage.length != rightimage.length) || (leftimage.length != rightimage.length))
        return;
    Boolean debug = false;
    int w = leftimage.length;
    int h = leftimage[0].length;
    Double d = (w * 0.2);
    int percent_no = d.intValue();
    double weight = 0.5;
    double half = 0.5;
```

```
    for (int i=0; i < percent_no; i++) {
        for (int j=0; j < h; j++) {

            int v = (int)(leftimage[w-i-1][j]*(weight) + rightimage[i][j]*(1-weight));
            if (v > 255)
                v = 255;
            else if (v < 0)
                v = 0;

            int v2 = (int)(rightimage[i][j]*(weight) + leftimage[w-i-1][j]*(1-weight));
            if (v2 > 255)
                v2 = 255;
            else if (v2 < 0)
                v2 = 0;
            leftimage[w-i-1][j] = v;
            rightimage[i][j] = v2;
            if (i >= percent_no - 1 && debug) {
                leftimage[w-i-1][j] = 0;
                rightimage[i][j] = 0;
            }

        }
        weight = half + half*((double)i/percent_no);
    }

}

public void transpose(int[][] m) {
```

```
    for (int i = 0; i < m.length; i++) {
        for (int j = i; j < m[0].length; j++) {
            int x = m[i][j];
            m[i][j] = m[j][i];
            m[j][i] = x;
        }
    }
}

public void swapRows(int[][] m) {
    for (int  i = 0, k = m.length - 1; i < k; ++i, --k) {
        int[] x = m[i];
        m[i] = m[k];
        m[k] = x;
    }
}

public void rotate90left(int[][] m) {
    transpose(m);
    swapRows(m);
}

public void rotate90right(int[][] m) {
    swapRows(m);
    transpose(m);
}

public static void main(String[] args) {
```

```
Image imageProcessor = new Image();
int[][] pixels = imageProcessor.GStoMat("54.gif");
int[][] pixels2 = imageProcessor.GStoMat("55.gif");

if (args.length == 0) {
    System.out.println("here");
    imageProcessor.ImageBlur(pixels, pixels2);
}
else if (args.length == 1) {
    try {
        int transposeno = Integer.parseInt(args[0]);
        if (transposeno < 0) {
            System.out.println("Negative number of transpositions detected: " + args[0]);
        }
        else {

            for (int i=0; i<transposeno; i++) {
                System.out.println("rotate");
                imageProcessor.rotate90right(pixels2);
                imageProcessor.rotate90right(pixels);
            }

            System.out.println("asd");
            imageProcessor.ImageBlur(pixels, pixels2);

            // rotate images back to original rotation
            for (int i=0; i<transposeno; i++) {
                System.out.println("unrotate");
                imageProcessor.rotate90left(pixels);
```

```
                imageProcessor.rotate90left(pixels2);
            }
        }
    } catch(NumberFormatException nfe) {
        System.out.println("Unable to parse parameter: " + args[0]);
    }

}
else {
    System.out.println("Unknown number of arguments: "+ args.length + ", argumets: " + args);
}

imageProcessor.matToGS(pixels, "check.gif");
imageProcessor.matToGS(pixels2, "check2.gif");
  }
}
```

Κεφάλαιο VI

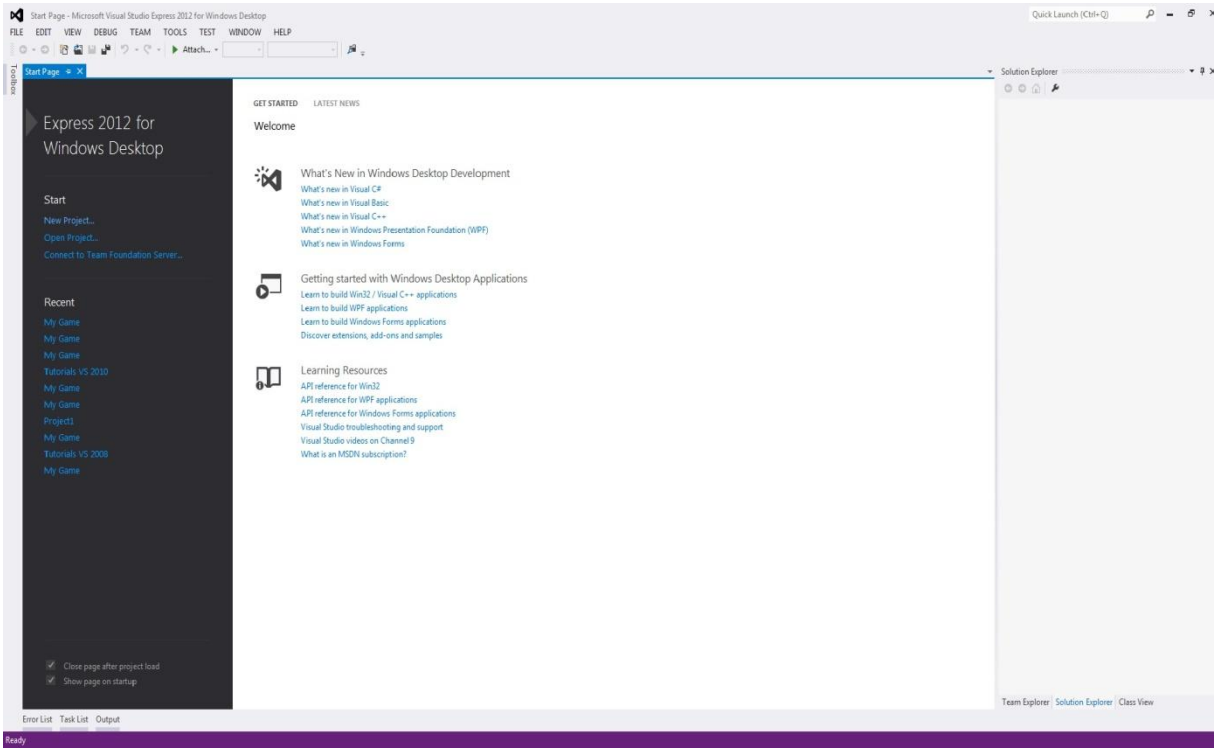
Σύνδεση με Visual Studio

6.1 Εισαγωγικά

Για να μπορέσει να ολοκληρωθεί η εργασία αυτή, ήταν αναγκαίο να χρησιμοποιήσουμε το Visual Studio της Microsoft και τις πολλές δυνατότητες που αυτό προσφέρει. Στο κεφάλαιο αυτό, θα δούμε αναλυτικά αυτό το περιβάλλον ανάπτυξης, θα δούμε το κώδικα που δημιουργήσαμε σε γλώσσα C++ καθώς και πως έγινε η διασύνδεση με τη μηχανή γραφικών που χρησιμοποιούμε έτσι ώστε να μπορέσει να ολοκληρωθεί η εργασία.

Πριν ξεκινήσουμε, όμως, θα πρέπει να έχουμε εγκαταστήσει στον υπολογιστή μας το Microsoft Visual Studio 2012. Εάν προσπαθήσετε να κατεβάσετε το περιβάλλον ανάπτυξης, θα διαπιστώσετε πως υπάρχουν πολλές διαφορετικές εκδόσεις. Μην σας ανησυχεί όμως αυτό, διότι και η απλή έκδοση καλύπτει απόλυτα για τη δουλειά που θέλουμε να γίνει, και μάλιστα διατίθεται δωρεάν.

Σε αυτή την εργασία χρησιμοποιήθηκε το Microsoft Visual Studio 2012 Ultimate Edition. Μπορούμε να το βρούμε (ή όποια άλλη έκδοση θέλουμε) στην εξής σελίδα: http://www.microsoft.com/visualstudio/eng/downloads. Μας δίνεται η δυνατότητα να κατεβάσουμε την έκδοση “Microsoft Visual Studio 2012 Express” η οποία διατίθεται δωρεάν και δεν υστερεί. Αφού κατεβάσουμε την έκδοση που επιθυμούμε και το εγκαταστήσουμε θα μας εμφανιστεί η αρχική οθόνη όπως φαίνεται στην παρακάτω εικόνα.



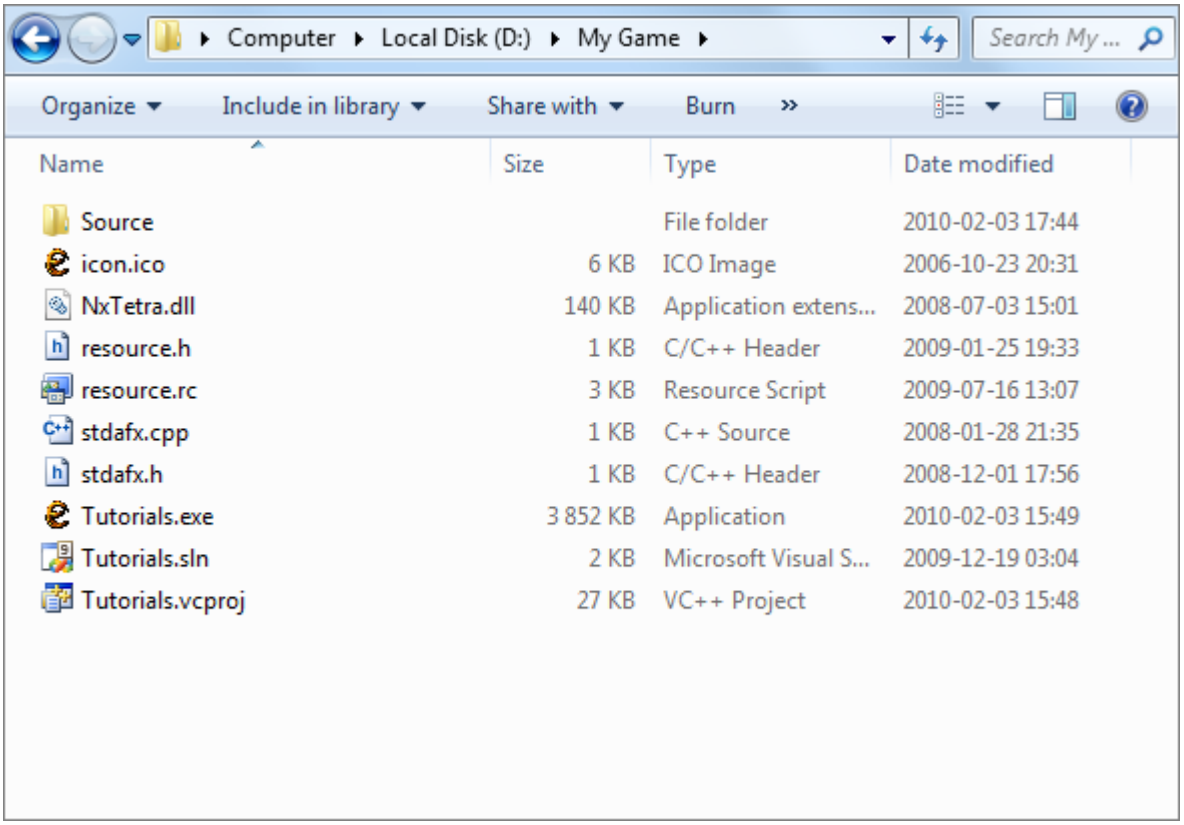
Εικόνα 19: Αρχική σελίδα του Microsoft Visual Studio Express 2012

Μετά την εγκατάσταση του Visual Studio, εμάς μας ενδιαφέρει να μπορέσουμε να συνδέσουμε και τη μηχανή γραφικών Esenthel έτσι ώστε να μπορούμε να κάνουμε τις αλλαγές που επιθυμούμε. Για να γίνει αυτό πρέπει να αντιγράψουμε κάποιες συγκεκριμένες βιβλιοθήκες (.lib) και επικεφαλίδες (headers) στα αρχεία του Visual Studio. Πρέπει να γίνουν δηλαδή τα ακόλουθα:

- Να αντιγραφεί το **"EsenthelEngineSDK\Installation\EsenthelEngine"** στο **"Microsoft Visual Studio\VC\include"**
- Να αντιγραφεί το **"EsenthelEngineSDK\Installation\EsenthelEngine.lib"** στο **"Microsoft Visual Studio\VC\lib"**
- Να αντιγραφεί το **"EsenthelEngineSDK\Installation\EsenthelEngineDX10+.lib"** στο **"Microsoft Visual Studio\VC\lib"**
- Να αντιγραφεί το **"EsenthelEngineSDK\Installation\EsenthelEngine64.lib"** στο **"Microsoft Visual Studio\VC\lib\amd64"**
- Να αντιγραφεί το **"EsenthelEngineSDK\Installation\EsenthelEngine64DX10+.lib"** στο **"Microsoft Visual Studio\VC\lib\amd64"**

6.2 Δημιουργία καινούριου project

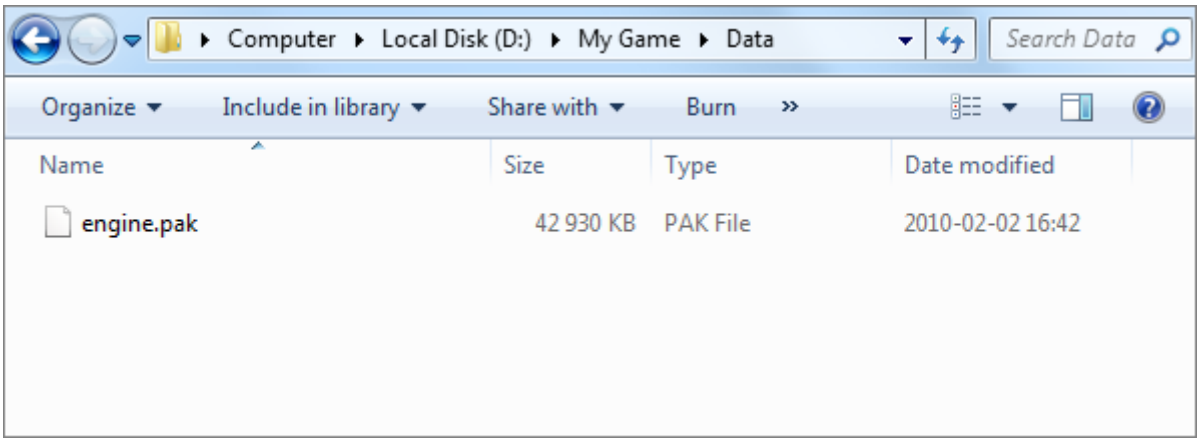
Ο ευκολότερος ουσιαστικά τρόπος για να δημιουργήσουμε ένα νεο project είναι να αντιγράψουμε σε κάποιο σημείο του δίσκου (έστω στο "D:\My Game") τον φάκελο "EsenthelEngineSDK\Tutorials" και να σβήσουμε όλα τα αρχεία που δεν πρόκειται να χρησιμοποιήσουμε.



Εικόνα 20: Σβήνουμε όλα τα αρχεία και κρατάμε μόνο τα περιεχόμενα του φακέλου "Source".

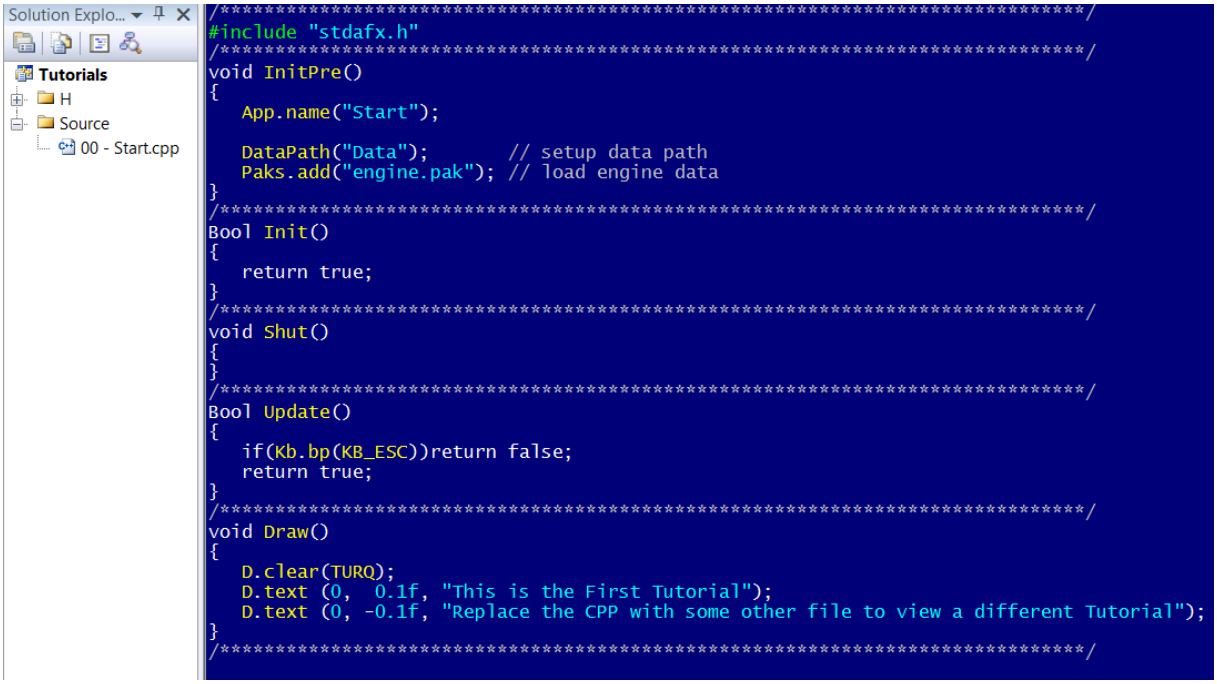
Στη συνέχεια και αφού έχουμε σβήσει οτιδήποτε δεν χρειαζόμαστε πρέπει να δημιουργήσουμε ένα φάκελο στον οποίο θα αποθηκεύουμε τα αρχεία που θα χρειαζόμαστε. Θα

ονομάσουμε αυτό το φάκελο “Data”. Επειδή η εφαρμογή ψάχνει να φορτώσει πρώτα απ’ όλα το "engine.pak" (ένα συμπιεσμένο αρχείο που περιέχει βασικές πληροφορίες για την εφαρμογή) το αντιγράφουμε μέσα στο φάκελο “Data”. Μπορούμε να το βρούμε μέσα στα αρχεία του EsenthelEngineSDK σε διάφορους φακέλους.



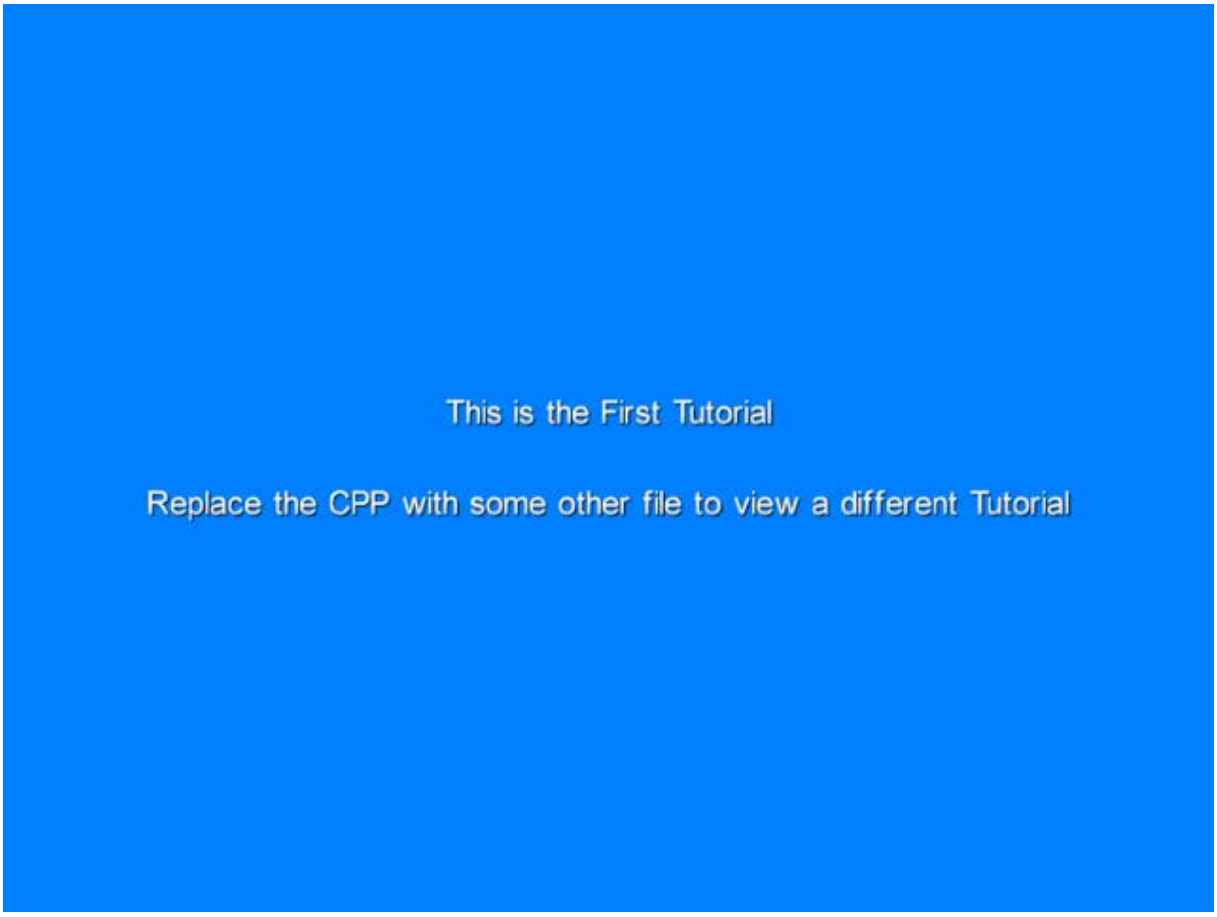
Εικόνα 21: engine.pak

Πλησιάζοντας σιγά σιγά στην ολοκλήρωση της δημιουργίας του project υπάρχουν μερικές παράμετροι ακόμα που πρέπει να ρυθμιστούν. Για να γίνει αυτό, τρέχουμε το αντιγραμμένο C++ project, τρέχοντας το "D:\My Game\Tutorials.sln". Βάζουμε το "D:\My Game\Source\00 - Start.cpp" να είναι το μοναδικό CPP αρχείο μέσα στο Solution Explorer και το τροποποιούμε έτσι ώστε να ξεκινά φορτώνοντας το path του φακέλου Data που δημιουργήσαμε και του “engine.pak” που τοποθετήσαμε μέσα. Φαίνονται καθαρά όλα αυτά στην εικόνα που ακολουθεί.



Εικόνα 22: Προσαρμόζωντας το Project για πρώτη φορά.

Έχοντας ολοκληρώσει τη διαδικασία και τρέχοντας το πρόγραμμα θα πρέπει να δούμε το εξής:



Εικονα 23: Εμφανίζεται στη πρώτη εκτέλεση του Project που δημιουργήσαμε

Αν και το project είναι έτοιμο προκειμένου να χρησιμοποιήσουμε και τον editor του Esenthel και να μπορούμε να επεξεργαζόμαστε και από κει πρέπει να γίνουν άλλα δυο βήματα.

- Αντιγράφουμε το "EsenthelEngineSDK\Data\Enum" στο "D:\My Game\Data\Enum"
- Και το "EsenthelEngineSDK\Data\Env" στο "D:\My Game\Data\Env"

6.3 Εναλλακτικά η δημιουργία του νέου Project

Η διαδικασία που αναλύσαμε πιο πάνω είναι ο εύκολος τρόπος δημιουργίας ενός καινούργιου project. Υπάρχει και η τυπική διαδικασία η οποία είναι λίγο πιο πολύπλοκη αλλά πρέπει να την αναφέρουμε.

Στο τρόπο αυτό πρέπει να δημιουργήσουμε ένα νέο Project στο Visual Studio και να τροποποιήσουμε αρκετά σημεία και τιμές σύμφωνα με αυτά που μας έχει πει ο κατασκευαστής. Η λίστα όλων των παραμέτρων που πρέπει να μεταβάλλουμε είναι αρκετά μεγάλη και είναι η παρακάτω:

C/C++

General

- Additional Include Directories = $(ProjectDir)
- Debug Information Format (**Debug, Win32**) = Program Database for Edit & Continue
- Debug Information Format (**Debug, x64**) = Program Database
- Debug Information Format (**Release**) = Disabled

Optimization

- Optimization (**Debug**) = Disabled
- Optimization (**Release**) = Maximize Speed
- Inline Function Expansion (**Debug**) = Default
- Inline Function Expansion (**Release**) = Any Suitable
- Enable Intrinsic Functions = Yes
- Favor Size or Speed = Favor Fast Code
- Omit Frame Pointers = Yes
- Enable Fiber-safe Optimizations = Yes

Preprocessor

- Preprocessor Definitions (**Debug**) = WIN32;DEBUG;_WINDOWS
- Preprocessor Definitions (**Release**) = WIN32;NDEBUG;_WINDOWS

Code generation

- Enable String Pooling = Yes
- Enable Minimal Rebuild (**Debug**) = Yes
- Enable Minimal Rebuild (**Release**) = No
- Basic Runtime Checks = Default
- Runtime Library = Multi-threaded
- Enable Enhanced Instruction Set (**Win32**) = Streaming SIMD Extensions 2
- Enable Enhanced Instruction Set (**x64**) = Not Set
- Floating Point Model = Fast

Language

- Default Char Unsigned = Yes
- Treat wchar_t as Built-in Type = Yes
- Enable Run-Time Type Info = Yes

Linker

General

- Output File = $(ProjectName).exe
- Enable Incremental Linking (**Debug**) = Yes
- Enable Incremental Linking (**Release**) = No

Input

- Additional Dependecies (**Win32, DX9, VS2008**) = EsenthelEngine.lib winmm.lib wininet.lib ws2_32.lib imm32.lib psapi.lib rpcrt4.lib version.lib libcmt.lib libcpmt.lib oldnames.lib iphlpapi.lib
- Additional Dependecies (**x64, DX9, VS2008**) = EsenthelEngine64.lib winmm.lib wininet.lib ws2_32.lib imm32.lib psapi.lib rpcrt4.lib version.lib libcmt.lib libcpmt.lib oldnames.lib iphlpapi.lib
- Additional Dependecies (**Win32, DX10+, VS2008**) = EsenthelEngineDX10+.lib winmm.lib wininet.lib ws2_32.lib imm32.lib psapi.lib rpcrt4.lib version.lib libcmt.lib libcpmt.lib oldnames.lib iphlpapi.lib
- Additional Dependecies (**x64, DX10+, VS2008**) = EsenthelEngine64DX10+.lib winmm.lib wininet.lib ws2_32.lib imm32.lib psapi.lib rpcrt4.lib version.lib libcmt.lib libcpmt.lib oldnames.lib iphlpapi.lib
- Additional Dependecies (**Win32, DX9, VS2010**) = EsenthelEngine.lib;winmm.lib;wininet.lib;ws2_32.lib;imm32.lib;psapi.lib;rpcrt4.lib;version.lib;libcmt.lib;libcpmt.lib;oldnames.lib;iphlpapi.lib;%(AdditionalDependencies)
- Additional Dependecies (**x64, DX9, VS2010**) = EsenthelEngine64.lib;winmm.lib;wininet.lib;ws2_32.lib;imm32.lib;psapi.lib;rpcrt4.lib;version.lib;libcmt.lib;libcpmt.lib;oldnames.lib;iphlpapi.lib;%(AdditionalDependencies)
- Additional Dependecies (**Win32, DX10+, VS2010**) = EsenthelEngineDX10+.lib;winmm.lib;wininet.lib;ws2_32.lib;imm32.lib;psapi.lib;rpcrt4.lib;version.lib;libcmt.lib;libcpmt.lib;oldnames.lib;iphlpapi.lib;%(AdditionalDependencies)
- Additional Dependecies (**x64, DX10+, VS2010**) = EsenthelEngine64DX10+.lib;winmm.lib;wininet.lib;ws2_32.lib;imm32.lib;psapi.lib;rpcrt4.lib;version.lib;libcmt.lib;libcpmt.lib;oldnames.lib;iphlpapi.lib;%(AdditionalDependencies)

- Ignore All Default Libraries = Yes

Debugging

- Generate Debug Info (**Debug**) = Yes
- Generate Debug Info (**Release**) = No

System

- Enable Large Addresses = Support Addresses Larger Than 2 Gigabytes

Optimization

- References (**Debug**) = Keep Unreferenced Data
- References (**Release**) = Eliminate Unreferenced Data
- Enable COMDAT Folding (**Debug**) = Do Not Remove Redundant COMDATs
- Enable COMDAT Folding (**Release**) = Remove Redundant COMDATs
- Link Time Code Generation = Default

Advanced

- Data Execution Prevention (DEP) = Image is not compatible with DEP

Command Line

- Additional Options = /IGNORE:4078

6.4 Τοποθέτηση αντικειμένων μέσα στον κόσμο

Ανατρέχοντας στα προηγούμενα κεφάλαια (βλ. Κεφάλαιο 5) έχουμε δημιουργήσει ήδη τους χάρτες για τον κόσμο μας και όπως είδαμε στο Κεφάλαιο 4 τους τοποθετούμε στο Esenthel στο Project που έχουμε δημιουργήσει. Οι αλλαγές που γίνονται στον Editor τώρα πια τροποποιούνται αυτόματα και στον κώδικα του Visual Studio.

Σκοπός εδώ όμως είναι να μπορέσουμε μέσα στο κόσμο που έχουμε δημιουργήσει να μπορέσουμε να προσθέσουμε μερικά αντικείμενα (objects) όπως για παράδειγμα δέντρα και βράχια. Για ένα αρκετά μεγάλο κόσμο όμως όπως είναι αυτός που δημιουργήσαμε είναι αρκετά χρονοβόρο και κουραστικό να τοποθετήσουμε αυτά τα αντικείμενα χειροκίνητα με βούρτσες ή άλλα εργαλεία που υπάρχουν στον Editor. Για να μπορέσουμε λοιπόν να διευκολυνθούμε θα πρέπει να δημιουργήσουμε στο Visual Studio μία συνάρτηση η οποία θα κάνει αυτή τη δουλειά για μας. Σαν πρώτη σκέψη μας έρχεται στο μυαλό ότι θα πρέπει να δίνουμε σαν παραμέτρους στη συνάρτηση πόσο συχνά πρέπει να τοποθετεί τα αντικείμενα και σε ποιο σημείο ακριβώς για να αποφύγουμε περιστατικά στα οποία το δέντρο ή οι πέτρες δεν θα αγγίζουν στο έδαφος. Για να το καταφέρουμε όμως αυτό πρέπει με κάποιο τρόπο να παίρνουμε τα στοιχεία του εδάφους σε κάθε σημείο στο οποίο θα τοποθετούμε ένα αντικείμενο. Αφού ο κόσμος μας είναι τρισδιάστατος σημαίνει ότι κάθε σημείο αποτελείται από τρεις αριθμούς, έναν για κάθε άξονα (x, y, z). Τη συνάρτηση αυτή την ονομάσαμε hmHeight. Παρακάτω ακολουθεί ο κώδικας σε C++ που γράψαμε στο Visual Studio:

```
/*****************************************************************************/
#include "stdafx.h" // include precompiled header
#include "../data/enum/_enums.h"
Game::ObjMemx<Game::Static> Statics; // container for static    objects
Game::ObjMemx<Game::Item  > Items;   // container for item      objects
Game::ObjMemx<Game::Chr   > Chrs;    // container for character objects
#define NT 2000
#define ARS 12
#define SNOW 100
#define SPEED 20
Mesh tree;
Vec2 Tpos[NT];
Flt mo = 0;


/*****************************************************************************/
void InitPre() // init before engine inits
{
   // here you will have to setup initial engine settings
   // like application name, its options, screen resolution...

   App.name("Start");// application name
   DataPath("../data");
   Paks.add("engine.pak"); // load engine data
   D.full(true).sync(true).hpRt(true).hdr(true);

   Cam.dist =0;
Cam.yaw =-PI_4;
Cam.pitch=-0.5;


Cam.at.set(16,0,16);
}

/*****************************************************************************/
Bool Init() // initialize after engine is ready
{
       Physics.create(CSS_NONE, true, "../Installation/PhysX");

   Sun.image="Env/Sky/sun.gfx";
   Sky.atmospheric();

   Game::World.init(); // initialize world, optionally you can change default
parameters here

   // once the world is initialized, we need to tell the world 'which class handles
which type'
   // this is done by assigning memory containers to certain Object Types defined in
Game Enums (which were used in the World Editor)
   Game::World.setObjType(Statics, OBJ_STATIC)  // set 'Statics' memory container for
'OBJ_STATIC' objects
              .setObjType(Items  , OBJ_ITEM  )  // set 'Items'   memory container for
'OBJ_ITEM'   objects
              .setObjType(Chrs   , OBJ_PLAYER); // set 'Chrs'    memory container for
'OBJ_PLAYER' objects

   // now when the engine is set up properly we can start a 'new game' with a builded
world
   Game::World.New("world/test1.world"); // create the world by giving path to builded
world
```

```
   // when the world is loaded it doesn't actually load all the terrain and objects
into memory
   // it loads only information about them
   // you need to tell the world which terrain and objects you need to use at the
moment
   // to do that call:
   Game::World.update(Cam.at); // which updates world to use only terrain and objects
at given position, here camera position is used
   // set initial camera
   Cam.setAngle(Vec(0,2,0), 0, -0.2f).set();
   for (int i = 0; i < ARS; i++)
for (int j = 0; j < ARS; j++)
{
int x = (i+0.5)*Game::World.areaSize();
int y = (j+0.5)*Game::World.areaSize();
Vec2 pos = Vec2(x, y);
Vec pos3 = Vec(x, 0, y);
Game::World.update(pos3);
Flt height = Game::World.hmHeight(pos);
height+= Game::World.hmHeight(Vec2(x-10, y));
height+= Game::World.hmHeight(Vec2(x+10, y));
height+= Game::World.hmHeight(Vec2(x, y-10));
height+= Game::World.hmHeight(Vec2(x, y+10));
height+= Game::World.hmHeight(Vec2(x-5, y));
height+= Game::World.hmHeight(Vec2(x+5, y));
height+= Game::World.hmHeight(Vec2(x, y-5));
height+= Game::World.hmHeight(Vec2(x, y+5));
height/=9;

if (height > SNOW)
{
// set overlay parameters
Flt overlay_angle =RandomF(PI2),
overlay_scale =40;
MaterialPtr overlay_material=Materials("Mtrl/Snow/snow.mtrl");

// add terrain overlay
Matrix m;
pos3.y = Game::World.hmHeight(pos);
m.setPosDir (pos3,Vec(0,1,0)); // set position and direction
m.rotateZVec(overlay_angle ); // rotate along matrix z axis
m.scaleOrn (overlay_scale ); // scale the overlay
Game::World.terrainAddOverlay(overlay_material,m, 1000);
}
}

// when the world is loaded it doesn't actually load all the terrain and objects into
memory
// it loads only information about them
// you need to tell the world which terrain and objects you need to use at the moment
// to do that call:
Cam.at.set(16,0,16);
Game::World.update(Cam.at); // which updates world to use only terrain and objects at
given position, here camera position is used
Cam.at.set(16,Game::World.hmHeight(Vec2(16,16)),16);
D.viewRange(1300);
Game::World.activeRange(1300);

Game::ObjParams &tree_obj=*Game::Objs("obj/static/tree1/0.obj");
Game::ObjParams &rock_obj=*Game::Objs("obj/static/rock/0.obj");
for (int i = 0; i < NT; i++)
{
```

```
Tpos[i] = Vec2(RandomF(ARS-1)*Game::World.areaSize(), RandomF(ARS-
1)*Game::World.areaSize());
if (i % 2 == 0)
Game::World.objCreate(tree_obj,Matrix(tree_obj.scale()/10,Vec(Tpos[i].x,Game::World.hm
Height(Vec2(Tpos[i].x,Tpos[i].y)-0.5, true)-0.5,Tpos[i].y))); // create new object at
(x,y,z) position and give objects default scaling
else
Game::World.objCreate(rock_obj,Matrix(rock_obj.scale(),Vec(Tpos[i].x,Game::World.hmHei
ght(Vec2(Tpos[i].x,Tpos[i].y), true)-0.5,Tpos[i].y))); // create new object at (x,y,z)
position and give objects default scaling
}


   return true;
}
/*****************************************************************************/
void Shut() // shut down at exit
{
   // this is called when the engine is about to exit
}
/*****************************************************************************/
Bool Update() // main updating
{

       if(Kb.bp(KB_ESC))return false;
   CamHandle(0.1f, 100, CAMH_ZOOM|(Ms.b(1)?CAMH_MOVE:CAMH_ROT));
   {
      if(Kb.b(KB_A    ))Cam.matrix.pos-=Cam.matrix.x*5*Time.d();
      if(Kb.b(KB_D    ))Cam.matrix.pos+=Cam.matrix.x*5*Time.d();
      if(Kb.b(KB_W    ))Cam.matrix.pos+=Cam.matrix.z*5*Time.d();
      if(Kb.b(KB_S    ))Cam.matrix.pos-=Cam.matrix.z*5*Time.d();
      if(Kb.b(KB_SPACE))Cam.matrix.pos+=Cam.matrix.y*5*Time.d();
      if(Kb.b(KB_LCTRL))Cam.matrix.pos-=Cam.matrix.y*5*Time.d();
      if(Ms.hidden())
      {
         Cam.yaw  -=Ms.d().x;
         Cam.pitch+=Ms.d().y;
      }
      Cam.setAngle(Cam.matrix.pos, Cam.yaw, Cam.pitch).updateVelocities().set();
   }
   Game::World.update(Cam.at); // update the world to given position

   return true;                    // continue


}
/*****************************************************************************/
void Render()
{
   Game::World.draw(); // draw world (this is done outside of 'switch(Renderer())'
because world automatically detects active rendering mode)
}
void Draw() // main drawing
{
   // here you have to tell engine what to draw on the screen

   Renderer(Render);

}

/*Game::World.hmHeight()*/
/*****************************************************************************/
```

Κεφάλαιο VII

Επίλογος και Μελλοντικές Προτάσεις

Μέσα σε αυτό το κείμενο παρουσιάστηκε η δουλειά η οποία έγινε για τη δημιουργία ενός σχετικά αυτοματοποιημένου τρόπου για τη κατασκευή κόσμων, αναλύθηκαν κάποιες βασικές μέθοδοι που χρησιμοποιήσαμε και έγινε προσπάθεια να έρθουν σε ένα χαμηλότερο επίπεδο έτσι ώστε να γίνουν κατανοητές από όλους. Βέβαια για κάθε μία διαφορετική σκέψη υπάρχει και διαφορετική υλοποίηση.

Κατά την προσωπική μας άποψη για να μπορέσει κάποιος να δημιουργήσει ένα κόσμο είτε μικρό είτε αρκετά μεγάλο όπως είναι ο δικός μας, χρειάζεται να είναι εφοδιασμένος με αρκετές γνώσεις προγραμματισμού και μαθηματικών αλλά και με γερά νεύρα. Τα παιχνίδια δεν είναι απλώς μια διαδικασία η οποία, αν μιλούσαμε στην γλώσσα του προγραμματισμού, έχει ένα σημείο εκκίνησης και ένα πέρας. Είναι μια επαναληπτική διαδικασία της οποίας οι μελλοντικές καταστάσεις εξαρτώνται άμεσα από τις προηγούμενες και αυτό είναι αρκετά δύσκολο για κάποιον που δεν το έχει ενστερνιστεί να το υλοποιήσει. Σίγουρα για τις γενιές που θα έρθουν θα υπάρχει μια σχετική ευκολία, διότι οι μηχανές γραφικών όσο περνάν τα χρόνια γίνονται όλο και πιο απλούστερες. Ήδη υπάρχουν περιβάλλοντα ανάπτυξης τα οποία είναι πολύ πιο εύχρηστα, πρέπει να πούμε όμως ότι και αυτά σε καμία περίπτωση δεν μπορούν να φτάσουν την προσαρμοστικότητα και επεκτασιμότητα των μηχανών γραφικών σε γραμμή εντολών.

Εξερευνώντας τον κόσμο των παιχνιδιών κάποιος διαπιστώνει ότι όσα πιο πολλά μαθαίνει και γνωρίζει, τόσο πιο πολύ αυξάνεται η πολυπλοκότητα. Αλλά πάντα η έμφυτη ανάγκη για εξερεύνηση του αγνώστου καθώς και η ανάγκη που βρίσκεται μέσα μας για δημιουργία μας φέρνει απέναντι σε φανταστικά αποτελέσματα.

Η εργασία αυτή μπορεί να δώσει κίνητρο ή να γίνει βάση για πλήθος άλλων εργασιών καθώς θα μπορούσε να αποτελέσει πηγή έμπνευσης ή εργαλείο δημιουργίας για κάποιον που θέλει να δημιουργήσει το δικό του κόσμο.

Η συγκεκριμένη εργασία μελλοντικά μπορεί να συνεχιστεί και να εξελιχτεί από κάποιον ενδιαφερόμενο, παίρνοντας σαν βάση τον ήδη υπάρχων κόσμο να προσπαθήσει να τον εμπλουτίσει με περισσότερα αντικείμενα, πέρα από δέντρα και βράχια βάζοντας από θάλασσες, ποτάμια και λίμνες μέχρι και πόλεις ολόκληρες. Σε ένα δεύτερο στάδιο είναι σχετικά απλό και φυσική συνέχεια η τοποθέτηση κάποιου ή κάποιων χαρακτήρων μέσα στο κόσμο χειριζόμενοι από χρήστες και ο κόσμος να ολοκληρωθεί. Και το παιχνίδι να αρχίσει…

Κεφάλαιο VIII

Βιβλιογραφία

1. Esenthel wiki. Διαθέσιμο: http://www.esenthel.com/wiki/index.php?title=Main_Page.
2. Esenthel documentation. Διαθέσιμο με την εγκατάσταση του προγράμματος.
3. C. Pozrikidis (2007): Introduction to C++ Programming and Graphics.
4. B. Chandra (2009): Object Oriented Programming Using C++.
5. Richard Johnsonbaugh, Martin Kalin (1995): Object Oriented Programming in C++.
6. Krysztof Czarnecki, Urlich Eisenecker (2000): Generative Programming.
7. Herbert Schildt (2004): The Art of C++.
8. David P. Luebke (2003): Level of Detail for 3D Graphics.
9. Alan Watt (1999): 3D Computer Graphics (3rd Edition).
10. Esenthel Forum. Διαθέσιμο: http://www.esenthel.com/community/.
11. Google. Διαθέσιμο: http://www.google.com.