Τεχνολογικό εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών

Τμήμα Εφαρμοσμένης Πληροφορικής & Πολυμέσων

Πτυχιακή εργασία

Τίτλος: Δημιουργία παιγνίου με χρήση XNA και Direct3D

Ποντικάκης Γεώργιος ΑΜ:2283

Abstract

This is a two-dimensional computer game made in XNA framework written in C#, using Direct3D11 HLSL shaders. It belongs to the category of arcade games containing rich visual and sound effects with integrated and easy to use menu. The purpose of this project is the research on the technologies that exist in this field, the study of real-time programming and the level of difficulty and complexity needed for its creation and its potential usage.

Σύνοψη

Πρόκειται για ένα διδιάστατο ηλεκτρονικό παιχνίδι φτιαγμένο στο XNA Framework σε γλώσσα C# που χρησιμοποιεί Direct3D11 shaders σε γλώσσα HLSL. Ανήκει στην κατηγορία των arcade παιχνιδιών περιέχοντας πλούσια οπτικά και ηχητικά effects με ολοκληρωμένο και εύχρηστο menu. Σκοπός της κατασκευής του είναι η έρευνα πάνω στις τεχνολογίες που υπάρχουν σε αυτόν τον τομέα, η μελέτη του προγραμματισμού σε πραγματικό χρόνο και το επίπεδο δυσκολίας και πολυπλοκότητας που χρειάζεται για την κατασκευή του, καθώς και οι δυνατότητες αξιοποίησης του.

Πίνακας περιεχομένων

Εισαγωγή

Όπως όλοι γνωρίζουμε, ο κόσμος των ηλεκτρονικών παιχνιδιών είναι μεγάλος και μεγαλώνει καθημερινά. Μαζί όμως με το κοινό και την ποικιλία του μεγαλώνουν και οι τεχνολογικές απαιτήσεις του από τους Η/Υ (και όχι μόνο). Καθώς οι κατασκευαστές παιχνιδιών προσπαθούν είτε να εξομοιώσουν τον πραγματικό κόσμο κάνοντας τα παιχνίδια όσο το δυνατόν ρεαλιστικότερα είτε να προσφέρουν θέαμα και εφέ, η ανάγκη για μεγαλύτερη επεξεργαστική δύναμη εμφανίζετε όλο και περισσότερο. Αυτό έχει φυσικά σαν αποτέλεσμα ένα τεράστιο κομμάτι της εξέλιξης της τεχνολογίας, ακόμα και από τις μεγαλύτερες εταιρίες στον χώρο να αποσκοπεί σε αυτό.

Λόγο της πολυπλοκότητας των αλγορίθμων και φυσικά του μεγέθους της δουλειάς που απαιτεί ένα μεγάλο παιχνίδι, εδώ και πολλά χρόνια οι μεγάλοι κατασκευαστές έχουν δημιουργήσει βασικές δομές στις οποίες πάνω χτίζουν τα παιχνίδια τους. Οι δύο γενικότερες κατηγορίες είναι οι μηχανές παιχνιδιών (game engines) και οι μηχανές γραφικών (graphic engines).

Το μεγάλο ερώτημα και ένας από τους λόγους αυτής της εργασίας είναι το τι τρόποι υπάρχουν που θα επιτρέψουν σε κάποιον να δημιουργήσει ένα δικό του παιχνίδι, χωρίς να έχει στην διάθεση του μια έτοιμη μηχανή παιχνιδιού η γραφικών και μέχρι που μπορεί να φτάσει.

Υπάρχουν διάφορα εργαλεία που μπορεί να χρησιμοποιήσει κανείς για να φτιάξει ένα ηλεκτρονικό παιχνίδι. Μερικά είναι δωρεάν και δεν χρειάζεται κανείς γνώσεις προγραμματισμού για να τα χρησιμοποιήσει αλλά οι δυνατότητες τους είναι περιορισμένες. Άλλα εργαλεία που μπορεί κανείς να αγοράσει σίγουρα έχουν μεγαλύτερες δυνατότητες αλλά έχουν την δική τους μηχανή παιχνιδιού και γραφικών. Μόνο ένα όμως δίνει την δυνατότητα της δημιουργίας ενός ηλεκτρονικού παιχνιδιού σχεδόν από το μηδέν, χωρίς την ανάγκη χρήσης έτοιμων μηχανών, σχεδόν χωρίς περιορισμούς και με πάρα πολλές δυνατότητες.

Αυτό είναι το XNA framework της Microsoft. Επέλεξα να το χρησιμοποιήσω για την κατασκευή του παιχνιδιού μου με σκοπό την μελέτη της τεχνολογίας των παιχνιδιών και των τρόπων αξιοποίησης του.

Μηχανή Παιχνιδιού

Μια μηχανή παιχνιδιού είναι ένα σύστημα λογισμικού σχεδιασμένο για τη δημιουργία και την ανάπτυξη βιντεοπαιχνιδιών. Υπάρχουν πολλές μηχανές παιχνιδιών οι οποίες είναι σχεδιασμένες να δουλεύουν σε κονσόλες βιντεοπαιχνιδιών και λειτουργικά συστήματα επιτραπέζιων υπολογιστών όπως τα Microsoft Windows, το Linux, και το Mac OS X. Η κεντρική λειτουργικότητα που παρέχεται τυπικά από μια μηχανή παιχνιδιού περιλαμβάνει μια μηχανή φωτοαπόδοσης ("renderer") για 2D ή 3D γραφικά, μια μηχανή φυσικής ή εντοπισμού συγκρούσεων (collision detection, καθώς και collision response), ήχο, scripting, animation, τεχνητή νοημοσύνη, δικτύωση, streaming, διαχείριση μνήμης, νήματα (threading), υποστήριξη τοπικοποίησης, και ένα γράφο σκηνής (scene graph). Η διαδικασία της ανάπτυξης παιχνιδιού συχνά οικονομικοποιείται με το ότι σε μεγάλο μέρος η ίδια μηχανή παιχνιδιού επαναχρησιμοποιείται για να δημιουργηθούν διαφορετικά παιχνίδια.

Λίστα γνωστότερων μηχανών παιχνιδιού:

- Unreal Engine
- Steam Engine
- CryEngine2
- Gamebryo
- Trinigy Vision Engine
- Multiverse Network
- Bigworld Technology
- HeroEngine
- Monumental Games
- Exit Games Neutron
- Project Darkstar
- DX Studio

Γραφικό Περιβάλλον

Τα γραφικά υπολογιστών ή απλώς γραφικά, είναι ένας επιστημονικός κλάδος της πληροφορικής που ασχολείται με τη θεωρία και την τεχνολογία αλγοριθμικής σύνθεσης εικόνων σε ηλεκτρονικό υπολογιστή. Κατά κανόνα, στα γραφικά υπολογιστών η είσοδος των υπολογισμών είναι συμβολικές περιγραφές της οπτικής σκηνής και η έξοδος ψηφιακές εικόνες ή βίντεο, με ή χωρίς αλληλεπίδραση με τον χρήστη. Τα γραφικά υπολογιστών έχουν μεγάλη ποικιλία εφαρμογών: αξιοποιούνται π.χ. στις γραφικές διεπαφές χρήστη (GUI), στα παιχνίδια υπολογιστών, στην εικονική πραγματικότητα και στην οπτικοποίηση δεδομένων. Επίσης εφαρμόζονται στην εικονογράφηση εντύπων, στην αρχιτεκτονική σχεδίαση, στη δημιουργία λογοτύπων αλλά και, ευρύτερα, στη δημιουργία ακριβέστατων σχεδίων.

Τύποι γραφικών υπολογιστών

Τα γραφικά υπολογιστών μπορούν να διακριθούν σε κατηγορίες, αναλόγως με κάποιο κριτήριο:

Με βάση το πλήθος των διαστάσεων οι οποίες συμμετέχουν στην απεικόνισή τους:

- Διδιάστατα (2D) γραφικά υπολογιστών
- Τρισδιάστατα (3D) γραφικά υπολογιστών

Με βάση τη χρονική στιγμή κατά την οποία λαμβάνει χώρα η απόδοσή τους (rendering):

- Στατικά γραφικά υπολογιστών
- Γραφικά υπολογιστών πραγματικού χρόνου

➢ Διδιάστατα (2D) γραφικά υπολογιστών

Τα διδιάστατα γραφικά υπολογιστών αποτελούν προσπάθειες απεικόνισης γραφικών δύο διαστάσεων στην οθόνη μιας ψηφιακής συσκευής (π.χ. ενός υπολογιστή). Συνήθως τέτοια γραφικά χρησιμοποιούνται για τη δημιουργία γραφικών διασυνδέσεων χρήστη (GUI), αλλά και για εικονογραφήσεις βιβλίων, περιοδικών και λοιπών εντύπων. Μετά τη σύνθεσή τους, αποθηκεύονται σε ψηφιακά αρχεία εικόνας και η περαιτέρω επεξεργασία τους παύει να αποτελεί αντικείμενο του πεδίου των γραφικών, απασχολώντας πλέον την ψηφιακή επεξεργασία εικόνας. Τα διδιάστατα γραφικά μπορούν να διαμεριστούν στις εξής δύο μεγάλες κατηγορίες:

Διανυσματικά γραφικά (Vector Graphics): Χρησιμοποιούνται για τη δημιουργία εικόνων όπως λογότυποι, σήματα κατατεθέντα κτλ. αλλά και ψευδοτριδιάστατων σχημάτων (προοπτική). Αποθηκεύονται ως μαθηματικοί τύποι αναλυτικής γεωμετρίας οι οποίοι περιγράφουν γεωμετρικά σχήματα, επομένως δεν ισχύει σε αυτά η έννοια της ανάλυσης εικόνας και μπορούν να μεγεθυνθούν / σμικρυνθούν χωρίς απώλεια ποιότητας.

Γραφικά ψηφίδων (Bitmap Graphics): Όλα τα γραφικά που δημιουργούνται από ψηφιοποίηση υπαρκτών αντικειμένων (φωτογραφίες, εικόνες από σαρωτές κτλ.) ανήκουν σε αυτή την κατηγορία. Τα γραφικά ψηφίδων λειτουργούν όπως ακριβώς ένα ψηφιδωτό: όσο μικρότερες και περισσότερες ψηφίδες χρησιμοποιούνται (μεγαλύτερη ανάλυση), τόσο πιο ευκρινές και ακριβές είναι το τελικό αποτέλεσμα. Η ανάλυση μετράται σε κουκκίδες (ψηφίδες) ανά ίντσα (dots per inch, dpi). Για μια οθόνη, η ανάλυση των 72 ή 96 dpi είναι επαρκέστατη, αν όμως η εικόνα προορίζεται για επαγγελματική εκτύπωση, το ελάχιστο απαιτούμενο είναι οι 300 dpi. Η απώλεια ποιότητας κατά τη μεγέθυνση μίας ψηφιογραφικής εικόνας είναι αντιστρόφως ανάλογη της ανάλυσής της. Επειδή οι οθόνες όπου τα γραφικά προβάλλονται χαρακτηρίζονται από τη δική τους ανάλυση, μία εικόνα με ανάλυση πολύ μικρότερη από της οθόνης πρέπει να

προβληθεί σε μικρές διαστάσεις για να μη φαίνεται θολή και χωρίς ευκρίνεια.

Ένα ακόμη χαρακτηριστικό των γραφικών είναι το βάθος χρώματος, δηλαδή το πλήθος των δυαδικών ψηφίων (bits) που χρησιμοποιούνται για την περιγραφή του χρώματος κάθε ψηφίδας (ή κάθε περιοχής στα διανυσματικά γραφικά). Το σημερινό πρότυπο για τις οθόνες είναι το βάθος χρώματος 24 bits, ενώ για τις εκτυπώσεις 32 bits (η οθόνη και η εκτύπωση χρησιμοποιούν διαφορετικά χρωματικά πρότυπα). Υπάρχουν και γραφικά με μεγαλύτερο βάθος χρώματος, που προορίζονται για ειδικές χρήσεις, καθώς το ανθρώπινο μάτι δεν μπορεί να διακρίνει περισσότερα από 16,7 εκατομμύρια χρωματικές διαβαθμίσεις.

Ο τύπος των γραφικών αναγνωρίζεται συνήθως από την επέκταση του ονόματος του αρχείου, στο οποίο είναι αποθηκευμένα (δηλ. το τμήμα εκείνο του ονόματος που βρίσκεται δεξιά από την τελεία που χωρίζει στα δύο το όνομα ενός αρχείου). Οι πλέον συνήθεις τύποι είναι:

- Διανυσματικά γραφικά: .svg, .cdr, .ai
- Γραφικά ψηφίδων: .tif, .bmp, jpg, .gif, .png

➢ Τριδιάστατα (3D) γραφικά υπολογιστών

Τα τριδιάστατα γραφικά υπολογιστών αποτελούν προσπάθειες απεικόνισης γραφικών τριών διαστάσεων στη - δύο διαστάσεων - οθόνη μιας ψηφιακής συσκευής (π.χ. ενός υπολογιστή). Το γεγονός ότι η απεικόνιση χρησιμοποιεί τρεις διαστάσεις τα καθιστά πολύ ρεαλιστικά. Η λειτουργία τους στηρίζεται στη χωρική περιγραφή τριδιάστατων αντικειμένων μέσω σημείων και μαθηματικών τύπων, σε κάποιο σύστημα συντεταγμένων, και στην προβολή ακολούθως των συντεταγμένων των σημείων τους σε δύο διαστάσεις κατά τη φάση της απόδοσης. Τέτοιου είδους γραφικά χρησιμοποιούνται συνήθως από προγράμματα όπως τα παιχνίδια υπολογιστών και οι εικονικοί κόσμοι. Τα τριδιάστατα γραφικά βρίσκουν επίσης εφαρμογή στον κινηματογράφο, για τη σύνθεση σκηνών εικονικών κόσμων

(χαρακτηριστικό παράδειγμα οι ταινίες του κύκλου Ο Άρχοντας των Δαχτυλιδιών) και για την υλοποίηση ειδικών εφέ μέσω της σύγχρονη ψηφιακής τεχνολογίας (αντί για μηχανικά ή προσθετικά εφέ).

➢ Στατικά γραφικά υπολογιστών

Τα στατικά γραφικά υπολογιστών αποτελούν προϋπολογισμένα και προεπεξεργασμένα αντικείμενα γραφικών (συντεταγμένες σημείων και επιφανειών, χρωματισμοί, φωτισμοί και υφές τους) τα οποία δεν αποδίδονται τη στιγμή που προβάλλονται, αλλά έχουν αποδοθεί μία φορά κατά τη δημιουργία τους. Στη συνέχεια αποθηκεύονται και αναπαράγονται υπό μορφή αρχείου βίντεο, επομένως δεν μπορούν να είναι αλληλεπιδραστικά. Παράδειγμα τέτοιων γραφικών είναι τα μικρά βίντεο, τα οποία εμφανίζονται π.χ. σε βιντεοπαιχνίδια, και τα οποία έχουν «γυριστεί» μια φορά και κάθε φορά που θα τα παρακολουθήσουμε παραμένουν ίδια. Για τη δημιουργία τους χρησιμοποιείται κάποιο κατάλληλο πρόγραμμα δημιουργίας γραφικών και κίνησης (animation) όπως το 3D Studio Max, το Maya, το Lightwave, το Blender, το Cinema4D κτλ.

➢ Γραφικά υπολογιστών πραγματικού χρόνου

Τα γραφικά υπολογιστών πραγματικού χρόνου είναι αντικείμενα γραφικών (συντεταγμένες σημείων και επιφανειών, χρωματισμοί, φωτισμοί και υφές τους) τα οποία αποδίδονται οπτικά κατά τη στιγμή που εκτελείται ένα πρόγραμμα υπολογιστή, κάθε φορά που αυτό συμβαίνει, με εκ νέου εκτέλεση των κατάλληλων εντολών / υπολογισμών από τον επεξεργαστή. Για παράδειγμα, τα γραφικά που εμφανίζονται στην οθόνη ενός υπολογιστή ο οποίος εκτελεί ένα βιντεοπαιχνίδι, ανήκουν συνήθως σε αυτήν την κατηγορία. Για την προβολή τους απαιτείται κάποια μηχανή γραφικών πραγματικού χρόνου, όπως για παράδειγμα η Ogre3D, η Irrlich και το Crystal Space.

Τα γραφικά πραγματικού χρόνου μπορούν να είναι και αλληλεπιδραστικά, με τη μηχανή γραφικών να αποκρίνεται κατάλληλα σε εισόδους του χρήστη (π.χ. από περιφερειακά όπως το ποντίκι ή το πληκτρολόγιο), αλλά αυτό δεν είναι απαραίτητο. Για τον προγραμματισμό τους υπάρχουν διάφορες προτυποποιημένες βιβλιοθήκες, όπως η OpenGL και το Direct3D.

Microsoft XNA

Το Microsoft XNA είναι ένα ιδιόκτητο δωρεάν σετ εργαλείων με ένα διαχειριζόμενο περιβάλλον εκτέλεσης που παρέχεται από τη Microsoft και διευκολύνει την ανάπτυξη και διαχείριση ηλεκτρονικών παιγνίων. Το XNA βασίζεται στο .NET Framework , με εκδόσεις για τα Windows, Windows Phone και το Xbox . Από πολλές απόψεις, το XNA μπορεί να θεωρηθεί ως ένα .NET ανάλογο με το πιο γνωστό σύστημα ανάπτυξης παιχνιδιών της Microsoft, το DirectX , αλλά απευθύνεται σε προγραμματιστές που ενδιαφέρονται για ελαφριά παιχνίδια που τρέχουν σε μια ποικιλία από πλατφόρμες της Microsoft. Το XNA είναι η βασική πλατφόρμα για το Xbox Live Indie Games .

Το XNA Framework ανακοινώθηκε 24 Μαρτίου 2004, στο συνέδριο Game Developers Conference στο Σαν Χοσέ, Καλιφόρνια . Η πρώτη τεχνολογική προεπισκόπηση του XNA κυκλοφόρησε στις 14 Μαρτίου, 2006. Το XNA Game Studio 2.0 κυκλοφόρησε τον Δεκέμβριο του 2007, ακολουθούμενο από το XNA Game Studio 3.0 στις 30 Οκτωβρίου, 2008. Το XNA Game Studio 4.0 κυκλοφόρησε στις 16 Σεπτεμβρίου 2010 μαζί με τα Windows Phone Development Tools. Σύμφωνα με ένα email που στάλθηκε στις 31 Ιανουαρίου του 2013, το XNA δεν είναι πια ενεργά αναπτυσσόμενο και δεν υποστηρίζεται στο πλαίσιο του νέου " Metro interface " των Windows 8.

Το Microsoft XNA Framework βασίζεται στο .NET Compact Framework 2.0 για το Xbox 360 και το .NET Framework 2.0 για Windows. Περιλαμβάνει ένα εκτεταμένο σύνολο class libraries, ειδικών για την ανάπτυξη παιχνιδιών, για να επιτευχθεί η μέγιστη δυνατή επαναχρησιμοποίηση κώδικα σε όλες τις πλατφόρμες. Το framework τρέχει σε μια έκδοση του Common Language Runtime που είναι βελτιστοποιημένη για gaming και παρέχει ένα διαχειριζόμενο περιβάλλον εκτέλεσης. Είναι διαθέσιμο για τα Windows XP , Windows Vista , Windows 7 , Windows Phone και Xbox 360.

Το XNA framework συμπυκνώνει χαμηλού επιπέδου τεχνολογικές λεπτομέρειες που εμπλέκονται στην κωδικοποίηση ενός παιχνιδιού,

βεβαιώνοντας ότι το ίδιο το πλαίσιο φροντίζει για τη διαφορά μεταξύ των πλατφορμών, όταν τα παιχνίδια μεταφερθεί από τη μία πλατφόρμα σε με μια άλλη, και με αυτόν τον τρόπο επιτρέποντας στους προγραμματιστές παιχνιδιών να επικεντρωθούν περισσότερο στο περιεχόμενο και την εμπειρία του gaming. Το XNA framework είναι ενσωματωμένο με μια σειρά από εργαλεία, όπως το Cross-platform Audio Creation Tool (XACT), για να βοηθήσει στη δημιουργία του παιχνιδιού. Παρέχει υποστήριξη για δημιουργία 2D και 3D παιχνιδιού και επιτρέπει τη χρήση Xbox 360 controllers. Παιχνίδια φτιαγμένα στο XNA framework που στοχεύουν την πλατφόρμα του Xbox μπορούν να διανέμονται μόνο από μέλη της Microsoft XNA Creator's Club, με μια συνδρομή 99 δολαρίων / έτος. Desktop εφαρμογές μπορούν να διανεμηθούν δωρεάν σύμφωνα με την ισχύουσα αδειοδότησης της Microsoft.

Το XNA Framework βοήθησε πολύ μικρές εταιρίες ή και απλά ομάδες ατόμων να κατασκευάσουν ηλεκτρονικά παιχνίδια. Μερικά από αυτά μάλιστα (όπως τα Terraria, Reus και Magicka) κατάφεραν να τραβήξουν τα βλέμματα πάνω τους με τεράστιους αριθμούς αγοραστών από όλο τον κόσμο.

Μπορεί στις αρχές του 2013 το XNA Framework να μην υποστηρίζεται πια από την Microsoft αλλά αυτό δεν είναι απαραίτητα και το τέλος του. Ένα πρόγραμμα με το όνομα Mono.XNA δημιουργήθηκε με σκοπό την μεταφορά του XNA στο open source-cross platform Mono framework. Στην συνέχεια αυτό είχε σαν αποτέλεσμα την δημιουργία του MonoGame. Ενός framework βασισμένο πάνω στο XNA Framework, με τις ίδιες δυνατότητες που όμως επεκτείνει το εύρος πλατφόρμων μιας και είναι συμβατό με iOS, Android συμπεριλαμβανομένου του OUYA, Mac OS X, Linux και Metro for Windows 8, Windows RT και Windows Phone 8, όπως επίσης και Playstation Mobile σε 2D.

MonoGame

Το MonoGame είναι ένα ελεύθερο λογισμικό που χρησιμοποιείται από τους σχεδιαστές παιχνιδιών με σκοπό να κάνουν τα Windows και Windows Phone παιχνίδια τους να τρέχουν σε άλλα συστήματα. Προς το παρόν υποστηρίζει τα Mac OS , Linux , iOS , Android , το PlayStation Mobile , και την κονσόλα Ouya. Υλοποιεί το Microsoft XNA 4 Application programming interface.

Το MonoGame επιχειρεί να εφαρμόσει πλήρως το XNA 4 API. Αυτό το επιτυγχάνει σε όλες τις πλατφόρμες Microsoft χρησιμοποιώντας SharpDX και DirectX. Όσο για τις μη Microsoft πλατφόρμες, οι ειδικές ικανότητες της πλατφόρμας δουλεύουν μέσω της βιβλιοθήκης OpenTK. Για τις πλατφόρμες OS X, iOS, και / ή το Android, το Xamarin platform runtime είναι απαραίτητο. Αυτό το runtime παρέχει μια συντονισμένη εφαρμογή OpenTK που επιτρέπει στην ομάδα MonoGame να επικεντρωθεί στην ρύθμιση των γραφικών της πλατφόρμας.

Οι δυνατότητες γραφικών του MonoGame προέρχονται από το OpenGL, OpenGL ES, ή το DirectX. Από την MonoGame έκδοση 3, το OpenGL 2 αποτέλεσε το επίκεντρο για τις δυνατότητες. Οι προηγούμενες εκδόσεις του MonoGame (2.5) χρησιμοποιούν OpenGL 1.x για την απόδοση των γραφικών. Αξιοποιώντας το OpenGL 2 επιτρέπει στο MonoGame να υποστηρίξει shaders με στόχο τις πιο προηγμένες δυνατότητες απόδοσης στην πλατφόρμα.

Το Content Management συνεχίζει να ακολουθεί το XNA 4 Content Manager. Η ομάδα MonoGame έχει δημιουργήσει ένα νέο περιεχόμενο δόμηση που μπορεί να ενσωματωθεί με το Microsoft Visual Studio για να παραδώσει Content building δυνατότητες για τα Windows 8 Desktop που οι Windows 7 χρήστες είχαν χρησιμοποιήσει στο Microsoft XNA.

Το MonoGame έχει χρησιμοποιηθεί για αρκετά εμπορικά επιτυχημένα παιχνίδια, συμπεριλαμβανομένων των Bastion, Transistor και Fez.

Direct3D

Το Direct3D είναι ένα graphic application programming interface (API) για τα Microsoft Windows . Χρησιμοποιείται για την απόδοση τρισδιάστατων γραφικών σε εφαρμογές όπου η αποδόσεις είναι σημαντικές, όπως τα παιχνίδια. Το Direct3D χρησιμοποιεί την επιτάχυνση υλικού , εάν είναι διαθέσιμη στην κάρτα γραφικών , επιτρέποντας την επιτάχυνση υλικού ολόκληρου του 3D rendering pipeline ή ακόμα και μερική επιτάχυνση. Το Direct3D εκθέτει τις προηγμένες δυνατότητες του 3D gpraphic hardware, συμπεριλαμβανομένων των Z-buffering , W-buffering, Stencil buffering , spatial anti-aliasing , alpha blending , το color blending, mipmapping , texture blending, clipping , culling , atmospheric effects, perspective correct texture mapping , προγραμματιζόμενα HLSL shaders και τα effects. Η ενσωμάτωση με άλλες DirectX τεχνολογίες επιτρέπει στο Direct3D να παραδώσει χαρακτηριστικά, όπως video mapping, hardware 3D rendering σε 2D overlay planes, και ακόμη και sprites, παρέχοντας τη χρήση των 2D και 3D γραφικά σε διαδραστικές σχέσεις των μέσων ενημέρωσης.

Το Direct3D περιέχει πολλές εντολές για 3D computer graphics rendering, Ωστόσο, από την έκδοση 8, Direct3D έχει αντικαταστήσει το DirectDraw framework και επίσης έχει πάρει την ευθύνη για την παροχή των 2D γραφικών . Η Microsoft προσπαθεί να ανανεώνει συνεχώς το Direct3D για να υποστηρίξει την τελευταία διαθέσιμη τεχνολογία για τις κάρτες γραφικών 3D. Το Direct3D προσφέρει πλήρη vertex software εξομοίωση, αλλά καθόλου pixel software εξομοίωση για χαρακτηριστικά που δεν είναι διαθέσιμα στο hardware. Για παράδειγμα, αν το πρόγραμμα που κάνει χρήση του Direct3D απαιτεί τεχνολογία pixel shader και η κάρτα γραφικών στον υπολογιστή του χρήστη δεν υποστηρίζει αυτό το χαρακτηριστικό, το Direct3D δεν θα το υπολογίσει. Το API δεν περιλαμβάνει Reference rasterizer (REF ή συσκευή), η οποία μιμείται μια γενική κάρτα γραφικών στο λογισμικό, αν και είναι πολύ αργά για τις περισσότερες εφαρμογές 3D σε πραγματικό χρόνο και συνήθως χρησιμοποιείται μόνο για τον

εντοπισμό σφαλμάτων. Ένα νέο real-time software rasterizer, WARP , έχει σχεδιαστεί για να μιμηθεί πλήρως το σύνολο των δυνατοτήτων του Direct3D 10.1 και περιλαμβάνεται με τα Windows 7 και τα Windows Vista Service Pack 2. Οι επιδόσεις του λέγεται ότι είναι στο ίδιο επίπεδο με τις low end 3D κάρτες σε multi-core επεξεργαστές.

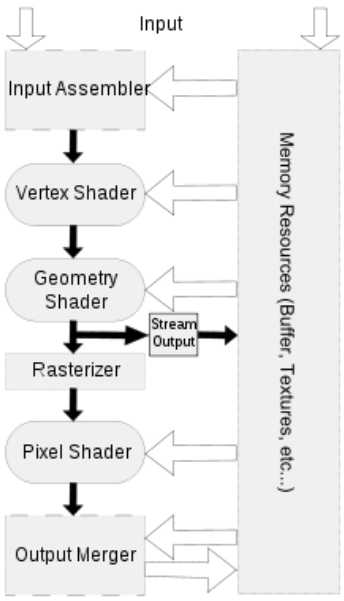
Μέρος του DirectX , το Direct3D είναι διαθέσιμο για τα Windows 95 και άνω, και είναι η βάση για το vector graphics API για τα Xbox και Xbox 360. Ο κύριος ανταγωνιστής του Direct3D είναι το OpenGL. Το Fahrenheit ήταν μια προσπάθεια από τις Microsoft και SGI για την ενοποιήσει του OpenGL και του Direct3D στη δεκαετία του 1990, αλλά τελικά ακυρώθηκε.

Direct3D 11

Το Direct3D 11 κυκλοφόρησε ως μέρος των Windows 7. Παρουσιάστηκε στο Gamefest 2008 στις 22 Ιουλίου 2008 και στο Nvision 08 τεχνικό συνέδριο στις 26 Αυγούστου, 2008. Η AMD παρουσίασε εργασία DirectX11 υλικού στο Computex στις 3 Ιουνίου 2009, το οποίο εκτελούσε ορισμένα DirectX 11 δείγματα του SDK. Τα χαρακτηριστικά του περιλαμβάνουν: Υποστήριξη για το Shader Model 5.0, dynamic shader linking, προσπελάσιμους πόρους, πρόσθετους τύπους πόρων, υπορουτίνες, geometry instancing, κάλυψη ως pixel shader εισόδου, προγραμματιζόμενη παρεμβολή των εισροών, νέες μορφές συμπίεσης textures (1 νέα μορφή LDR και 1 νέα μορφή HDR), texture clamps για τον περιορισμό προφόρτισης του WDD, 16K texture όρια, Gather4 (υποστήριξη για multi-component textures, υποστήριξη για προγραμματιζόμενα offsets), DrawIndirect, συντηρητική oDepth, Depth Bias, προσπελάσιμη ροή εξόδου, per-resource mipmap clamping, floating-point παράθυρα, οι εντολές μετατροπής shader, βελτιωμένη multithreading.

Direct3D Pipeline

1. **Input Assembler**: Διαβάζει σε vertex τα δεδομένα από μια εφαρμογή και τα προωθεί στο Pipeline.
2. **Vertex Shader**: Εκτελεί λειτουργίες σε ένα μόνο vertex την φορά.
3. **Hull Shader**: Εκτελεί λειτουργίες σε σύνολα patch control points, και δημιουργεί επιπλέον στοιχεία γνωστά ως patch constants.
4. **Tesselation stage**: Υποδιαιρεί τη γεωμετρία για να δημιουργήσει high-order αναπαραστάσεις του hull.
5. **Domain Shader**: Εκτελεί λειτουργίες στις εξόδους του tesselation, με τον ίδιο τρόπο όπως το vertex shader.
6. **Geometry Shader**: επεξεργάζεται ολόκληρα primatives όπως τρίγωνα, σημεία ή γραμμές.
7. **Stream Output**: Μπορεί να καταγράψει τα αποτελέσματα των προηγούμενων βημάτων στην μνήμη.
8. **Rasterizer**: Μετατρέπει primatives σε pixels και τα περνάει στον pixel shader.
9. **Pixel Shader**: Καθορίζει το τελικό χρώμα του pixel για να γραφτεί με στον render target και μπορεί επίσης να υπολογίσει μια τιμή βάθους για να γραφτεί στον depth buffer.
10. **Output Merger**: Συγχωνεύει διαφόρων τύπων δεδομένα εξόδου για την κατασκευή του τελικού αποτελέσματος.



High-Level Shading Language (HLSL)

Η High-Level Shading Language (HLSL) είναι μια ιδιόκτητη γλώσσα shading που κατασκευάστηκε από τη Microsoft για χρήση με το Microsoft Direct3D API. Είναι ανάλογη με την GLSL γλώσσα shading που χρησιμοποιείται με το OpenGL πρότυπο. Είναι πολύ παρόμοια με τη Nvidia Cg shading language, όπως αναπτύχθηκαν παράλληλα.

Προγράμματα HLSL έρχονται σε πέντε μορφές: pixel shader, vertex shader, geometry shader, compute shader και tessellation shader. Ο vertex shader εκτελείται για κάθε κορυφή που έχει υποβληθεί από την εφαρμογή, και είναι κυρίως υπεύθυνος για τη μετατροπή της κορυφής από τον χώρου του αντικείμενο στον οπτικό χώρο, δημιουργώντας συντεταγμένες για το texture, και τον υπολογισμό των συντελεστών φωτισμού, όπως το vertex tangent, binormal και normal vectors. Όταν μια ομάδα από vertices (συνήθως 3, για να σχηματίσουν ένα τρίγωνο) περάσει από τον vertex shader, η θέση τους παρεμβάλετε για να σχηματίσουν pixels εντός της περιοχής και αυτή η διαδικασία είναι γνωστή ως rasterisation . Κάθε ένα από αυτά τα pixels έρχεται μέσα από την τεχνολογία pixel shader, σύμφωνα με την οποία υπολογίζεται το τελικό χρώμα στην οθόνη.

Προαιρετικά, μια εφαρμογή που χρησιμοποιεί μια διεπαφή Direct3D 10/11 και 10/11 Direct3D υλικό μπορεί επίσης να καθορίσει ένα geometry shader. Αυτό το shader παίρνει ως είσοδο του τις τρεις κορυφές ενός τριγώνου και χρησιμοποιεί αυτά τα δεδομένα για να δημιουργήσει επιπλέον τρίγωνα, καθένα από τα οποία στη συνέχεια διαβιβάζεται στον rasterizer.

Σύγκριση των Shader Model

Σύγκριση των Pixel Shader

- PS 2.0 = DirectX 9.0 original Shader Model 2 specification.

- PS 2.0a = NVIDIA GeForce FX/PCX-optimized model.
- PS 2.0b = ATI Radeon X700, X800, X850, FireGL X3-256, V5000, V5100 and V7100 shader model, DirectX 9.0b.
- PS 3.0 = Shader Model 3.0.
- PS 4.0 = Shader Model 4.0.
- PS 4.1 = Shader Model 4.1.
- PS 5.0 = Shader Model 5.0.

Σύγκριση των Vertex Shader

- VS 2.0 = DirectX 9.0 original Shader Model 2 specification.
- VS 2.0a = NVIDIA GeForce FX/PCX-optimized model.
- VS 3.0 = Shader Model 3.0.
- VS 4.0 = Shader Model 4.0.
- VS 4.1 = Shader Model 4.1.
- VS 5.0 = Shader Model 5.0.

Αντικειμενοστρεφής Προγραμματισμός

Στην πληροφορική αντικειμενοστρεφή προγραμματισμό (object-oriented programming), ή ΑΠ, ονομάζουμε ένα προγραμματιστικό υπόδειγμα το οποίο εμφανίστηκε στα τέλη της δεκαετίας του 1960 και καθιερώθηκε κατά τη δεκαετία του 1990, αντικαθιστώντας σε μεγάλο βαθμό το παραδοσιακό υπόδειγμα του δομημένου προγραμματισμού. Πρόκειται για μία μεθοδολογία ανάπτυξης προγραμμάτων, υποστηριζόμενη από κατάλληλες γλώσσες προγραμματισμού, όπου ο χειρισμός σχετιζόμενων δεδομένων και των διαδικασιών που επενεργούν σε αυτά γίνεται από κοινού, μέσω μίας δομής δεδομένων που τα περιβάλλει ως αυτόνομη οντότητα με ταυτότητα και δικά της χαρακτηριστικά. Αυτή η δομή δεδομένων καλείται αντικείμενο και αποτελεί πραγματικό στιγμιότυπο στη μνήμη ενός σύνθετου, και πιθανώς οριζόμενου από τον χρήστη, τύπου δεδομένων ονόματι κλάση. Η κλάση προδιαγράφει τόσο δεδομένα όσο και τις διαδικασίες οι οποίες επιδρούν επάνω τους· αυτή υπήρξε η πρωταρχική καινοτομία του ΑΠ.

Έτσι μπορεί να οριστεί μία προδιαγραφή δομής αποθήκευσης (π.χ. μία κλάση «τηλεόραση») η οποία να περιέχει τόσο ιδιότητες (π.χ. μία μεταβλητή «τρέχον κανάλι») όσο και πράξεις ή χειρισμούς επί αυτών των ιδιοτήτων (π.χ. μία διαδικασία «άνοιγμα της τηλεόρασης»). Στο εν λόγω παράδειγμα κάθε υλική τηλεόραση (κάθε αντικείμενο αποθηκευμένο πραγματικά στη μνήμη) αναπαρίσταται ως ξεχωριστό, «φυσικό» στιγμιότυπο αυτής της πρότυπης, ιδεατής κλάσης. Επομένως μόνο τα αντικείμενα καταλαμβάνουν χώρο στη μνήμη του υπολογιστή ενώ οι κλάσεις αποτελούν απλώς «καλούπια». Οι αιτίες που ώθησαν στην ανάπτυξη του ΑΠ ήταν οι ίδιες με αυτές που οδήγησαν στην ανάπτυξη του δομημένου προγραμματισμού (ευκολία συντήρησης, οργάνωσης, χειρισμού και επαναχρησιμοποίησης κώδικα μεγάλων και πολύπλοκων εφαρμογών), όμως τελικώς η αντικειμενοστρέφεια επικράτησε καθώς μπορούσε να αντεπεξέλθει σε προγράμματα πολύ μεγαλύτερου όγκου και πολυπλοκότητας.

Έννοιες

Κεντρική ιδέα στον αντικειμενοστρεφή προγραμματισμό είναι η **κλάση** (**class**), μία αυτοτελής και αφαιρετική αναπαράσταση κάποιας κατηγορίας αντικειμένων, είτε φυσικών αντικειμένων του πραγματικού κόσμου είτε νοητών, εννοιολογικών αντικειμένων, σε ένα περιβάλλον προγραμματισμού. Πρακτικώς είναι ένας τύπος δεδομένων, ή αλλιώς το προσχέδιο μίας δομής δεδομένων με δικά της περιεχόμενα, τόσο μεταβλητές όσο και διαδικασίες. Τα περιεχόμενα αυτά δηλώνονται είτε ως δημόσια (public) είτε ως ιδιωτικά (private), με τα ιδιωτικά να μην είναι προσπελάσιμα από κώδικα εκτός της κλάσης. Οι διαδικασίες των κλάσεων συνήθως καλούνται μέθοδοι (methods) και οι μεταβλητές τους γνωρίσματα (attributes) ή πεδία (fields). Μία κλάση πρέπει ιδανικά να είναι εννοιολογικά αυτοτελής, να περιέχει δηλαδή μόνο πεδία τα οποία περιγράφουν μία κατηγορία αντικειμένων και δημόσιες μεθόδους οι οποίες επενεργούν σε αυτά όταν καλούνται από το εξωτερικό πρόγραμμα, χωρίς να εξαρτώνται από άλλα δεδομένα ή κώδικα εκτός της κλάσης, και επαναχρησιμοποιήσιμη, να αποτελεί δηλαδή μαύρο κουτί δυνάμενο να λειτουργήσει χωρίς τροποποιήσεις ως τμήμα διαφορετικών προγραμμάτων.

Αντικείμενο (object)

Αντικείμενο (object) είναι το στιγμιότυπο μίας κλάσης, δηλαδή αυτή καθαυτή η δομή δεδομένων (με αποκλειστικά δεσμευμένο χώρο στη μνήμη) βασισμένη στο «καλούπι» που προσφέρει η κλάση. Παραδείγματος χάρη, σε μία αντικειμενοστρεφή γλώσσα προγραμματισμού θα μπορούσαμε να ορίσουμε κάποια κλάση ονόματι BankAccount, η οποία αναπαριστά έναν τραπεζικό λογαριασμό, και να δηλώσουμε ένα αντικείμενο της με όνομα MyAccount. Το αντικείμενο αυτό θα έχει δεσμεύσει χώρο στη μνήμη με βάση τις μεταβλητές και τις μεθόδους που περιγράψαμε όταν δηλώσαμε την κλάση. Έτσι, στο αντικείμενο θα μπορούσε να περιέχεται ένα γνώρισμα Balance (=υπόλοιπο) και μία μέθοδος GetBalance (=επέστρεψε το υπόλοιπο).

Ακολούθως θα μπορούσαμε να δημιουργήσουμε ακόμα ένα ή περισσότερα αντικείμενα της ίδιας κλάσης τα οποία θα είναι διαφορετικές δομές δεδομένων (διαφορετικοί τραπεζικοί λογαριασμοί στο παράδειγμα). Τα αντικείμενα μίας κλάσης μπορούν να προσπελάσουν τα ιδιωτικά περιεχόμενα άλλων αντικειμένων της ίδιας κλάσης.

Ενθυλάκωση δεδομένων (data encapsulation)

Ενθυλάκωση δεδομένων (data encapsulation) καλείται η ιδιότητα που προσφέρουν οι κλάσεις να «κρύβουν» τα ιδιωτικά δεδομένα τους από το υπόλοιπο πρόγραμμα και να εξασφαλίζουν πως μόνο μέσω των δημόσιων μεθόδων τους θα μπορούν αυτά να προσπελαστούν. Αυτή η τακτική παρουσιάζει μόνο οφέλη καθώς εξαναγκάζει κάθε εξωτερικό πρόγραμμα να φιλτράρει το χειρισμό που επιθυμεί να κάνει στα πεδία μίας κλάσης μέσω των ελέγχων που μπορούν να περιέχονται στις δημόσιες μεθόδους της κλάσης.

Αφαίρεση δεδομένων

Αφαίρεση δεδομένων καλείται η ιδιότητα των κλάσεων να αναπαριστούν αφαιρετικά πολύπλοκες οντότητες στο προγραμματιστικό περιβάλλον. Μία κλάση αποτελεί ένα αφαιρετικό μοντέλο κάποιας κατηγορίας αντικειμένων. Επίσης οι κλάσεις προσφέρουν και αφαίρεση ως προς τον υπολογιστή, εφόσον η καθεμία μπορεί να θεωρηθεί ένας μικρός και αυτάρκης υπολογιστής (με δική του κατάσταση, μεθόδους και μεταβλητές).

Κληρονομικότητα

Κληρονομικότητα ονομάζεται η ιδιότητα των κλάσεων να επεκτείνονται σε νέες κλάσεις, ρητά δηλωμένες ως κληρονόμους (υποκλάσεις ή 'θυγατρικές κλάσεις'), οι οποίες μπορούν να επαναχρησιμοποιήσουν τις μεταβιβάσιμες μεθόδους και ιδιότητες της γονικής τους κλάσης αλλά και να προσθέσουν δικές τους. Στιγμιότυπα των θυγατρικών κλάσεων

μπορούν να χρησιμοποιηθούν όπου απαιτούνται στιγμιότυπα των γονικών (εφόσον η θυγατρική είναι κατά κάποιον τρόπο μία πιο εξειδικευμένη εκδοχή της γονικής), αλλά το αντίστροφο δεν ισχύει. Παράδειγμα κληρονομικότητας είναι μία γονική κλάση Vehicle (=Όχημα) και οι δύο πιο εξειδικευμένες υποκλάσεις της Car (=Αυτοκίνητο) και Bicycle (=Ποδήλατο), οι οποίες λέμε ότι "κληρονομούν" από αυτήν. Πολλαπλή κληρονομικότητα είναι η δυνατότητα που προσφέρουν ορισμένες γλώσσες προγραμματισμού μία κλάση να κληρονομεί ταυτόχρονα από περισσότερες από μία γονικές. Από μία υποκλάση μπορούν να προκύψουν νέες υποκλάσεις που κληρονομούν από αυτήν, με αποτέλεσμα μία ιεραρχία κλάσεων που συνδέονται μεταξύ τους "ανά γενιά" με σχέσεις κληρονομικότητας.

Υπερφόρτωση μεθόδου (method overloading)

Υπερφόρτωση μεθόδου (method overloading) είναι η κατάσταση κατά την οποία υπάρχουν, στην ίδια ή σε διαφορετικές κλάσεις, μέθοδοι με το ίδιο όνομα και πιθανώς διαφορετικά ορίσματα. Αν πρόκειται για μεθόδους της ίδιας κλάσης διαφοροποιούνται μόνο από τις διαφορές τους στα ορίσματα και στον τύπο επιστροφής.

Υποσκέλιση μεθόδου (method overriding)

Υποσκέλιση μεθόδου (method overriding) είναι η κατάσταση κατά την οποία μία θυγατρική κλάση και η γονική της έχουν μία μέθοδο ομώνυμη και με τα ίδια ορίσματα. Χάρη στη δυνατότητα του πολυμορφισμού ο μεταγλωττιστής «ξέρει» πότε να καλέσει ποια μέθοδο, βασισμένος στον τύπο του τρέχοντος αντικειμένου. Δηλαδή πολυμορφισμός είναι η δυνατότητα των αντικειμενοστρεφών μεταγλωττιστών να αποφασίζουν δυναμικά ποια είναι η κατάλληλη να κληθεί μέθοδος σε συνθήκες υποσκέλισης.

Αφηρημένη κλάση (abstract class)

Αφηρημένη κλάση (abstract class) είναι μία κλάση που ορίζεται μόνο για να κληρονομηθεί σε θυγατρικές υποκλάσεις και δεν υπάρχουν δικά της στιγμιότυπα (αντικείμενα). Η αφηρημένη κλάση ορίζει απλώς ένα "συμβόλαιο" το οποίο θα πρέπει να ακολουθούν οι υποκλάσεις της όσον αφορά τις υπογραφές των μεθόδων τους (όπου ως υπογραφή ορίζεται το όνομα, τα ορίσματα και η τιμή επιστροφής μίας διαδικασίας). Μία αφηρημένη κλάση μπορεί να έχει και μη αφηρημένες μεθόδους οι οποίες υλοποιούνται στην ίδια την κλάση (αν και φυσικά μπορούν να υποσκελίζονται σε υποκλάσεις). Αντιθέτως οι αφηρημένες μέθοδοί της είναι απλώς ένας ορισμός της υπογραφής τους και εναπόκειται στις υποκλάσεις να τις υλοποιήσουν. Μία αφηρημένη κλάση που δεν έχει γνωρίσματα και όλες οι μέθοδοί της είναι αφηρημένες και δημόσιες καλείται διασύνδεση (interface). Οι κλάσεις που κληρονομούν από μία διασύνδεση λέγεται ότι την "υλοποιούν".

C Sharp (C#) Γλώσσα Προγραμματισμού

Η C # είναι ένα multi-paradigm γλώσσα προγραμματισμού που περιλαμβάνει strong typing , imperative , declarative , functional , generic , object-oriented, και component-oriented ιδιότητες προγραμματισμού. Αναπτύχθηκε από τη Microsoft στο .NET Framework και αργότερα εγκρίθηκε ως πρότυπο από την Ecma (ECMA-334) και το ISO (ISO / IEC 23270: 2006). Η C # είναι μία από τις γλώσσες προγραμματισμού σχεδιασμένη για την Common Language Infrastructure.

Στόχοι σχεδιασμού

- Η γλώσσα C # προορίζεται να είναι μια απλή, σύγχρονη, γενικής χρήσης, αντικειμενοστραφής γλώσσα προγραμματισμού.
- Η γλώσσα, και οι εφαρμογές της, θα πρέπει να παρέχουν στήριξη για τις αρχές του μηχανικού λογισμικού, όπως strong type ελέγχου, σειρά ορίων ελέγχου , ελέγχου ορίου πινάκων, και αυτόματο garbage collector.
- Η γλώσσα προορίζεται για χρήση στην ανάπτυξη στοιχείων λογισμικού κατάλληλα για εφαρμογή σε κατανεμημένα περιβάλλοντα.
- Η φορητότητα του πηγαίου κώδικας είναι πολύ σημαντική, όπως είναι η φορητότητα του προγραμματιστής, ειδικά για εκείνους τους προγραμματιστές που είναι ήδη εξοικειωμένοι με C και C ++.
- Η Υποστήριξη για τη διεθνοποίηση είναι πολύ σημαντική.
- Η C # προορίζεται να είναι κατάλληλη για τη σύνταξη εφαρμογών για φιλοξενούμενα και ενσωματωμένα συστήματα.
- Αν και η C # προορίζεται να είναι οικονομική σε σχέση με τη μνήμη και την επεξεργαστική ισχύ, η γλώσσα δεν απέβλεπε στο να ανταγωνιστεί άμεσα τις επιδόσεις της C.

Real Time Computing

Στην επιστήμη των υπολογιστών , real time computing (RTC), ή reactive Computing , είναι η μελέτη των υλικών και των λογισμικών συστημάτων που υπόκεινται σε « περιορισμούς πραγματικού χρόνου», για παράδειγμα, οι προθεσμίες γεγονότων σε απόκριση του συστήματος. Προγράμματα σε πραγματικό χρόνο πρέπει να εγγυάται απάντηση εντός αυστηρών χρονικών περιορισμών, που συχνά αναφέρεται ως «deadlines». Απαντήσεις σε πραγματικό χρόνο συχνά γίνεται κατανοητό να είναι της τάξης των χιλιοστών του δευτερολέπτου, και μερικές φορές μικροδευτερόλεπτα. Αντίθετα, ένα σύστημα χωρίς εγκαταστάσεις σε πραγματικό χρόνο, δεν μπορεί να εγγυηθεί μια απάντηση σε οποιοδήποτε χρονικό διάστημα (ανεξάρτητα από την πραγματική ή αναμενόμενη χρόνους απόκρισης).

Real Time Computer Graphics

Real Time Computer Graphics είναι ένα υποπεδίο των computer graphics που εστιάζεται στην παραγωγή και την ανάλυση των εικόνων σε πραγματικό χρόνο . Ο όρος πιο συχνά χρησιμοποιείται σε διαδραστικά 3D γραφικά υπολογιστών , χρησιμοποιώντας συνήθως μια GPU, κάτι που απασχολεί κυρίως τον κόσμο των video games. Ο όρος μπορεί επίσης να παραπέμπει σε κάτι όπως το rendering ενός GUI μιας εφαρμογής, το real time image processing και την ανάλυση μιας εικόνας.

Παρά το γεγονός ότι οι υπολογιστές μπορούν από την αρχή να είναι σε θέση να παράγουν εικόνες 2D που αφορούν απλές γραμμές, εικόνες και πολύγωνα σε πραγματικό χρόνο, η δημιουργία 3D γραφικών του υπολογιστή και η ταχύτητα που είναι απαραίτητη για τη δημιουργία γρήγορης, καλής ποιότητας 3D εικόνες σε μια οθόνη ήταν πάντα ένα δύσκολο έργο για την παραδοσιακή αρχιτεκτονική Von Neumann architecture.

Βασικές Έννοιες και Χαρακτηριστικά του XNA

Κατά την δημιουργία ενός νέου XNA project υπάρχουν πέντε βασικές override methods. Όλη η δομή του παιχνιδιού βασίζετε σε αυτές τις μεθόδους για την κατασκευή και την εκτέλεση του.

- Protected override void Initialize()
- Protected override void LoadContent()
- Protected override void UnloadContent()
- Protected override void Update(GameTime gameTime)
- Protected override void Draw(GameTime gameTime)

Το ίδιο το XNA Framework φροντίζει για την διαχείριση τους και της καλεί με την ακόλουθη σειρά κατά την εκτέλεση του προγράμματος.

1. Initialize
2. LoadContent
3. Update
4. Draw

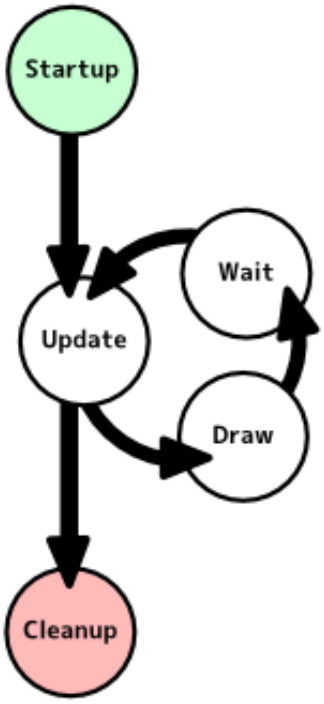
Η μέθοδος UnloadContent καλείτε στο τέλος της λειτουργίας του προγράμματος. Μετά από την επιτυχημένη εκτέλεση των παραπάνω μεθόδων το πρόγραμμα θα επαναλαμβάνει την εκτέλεση της Update και Draw με αποτέλεσμα την δημιουργία μιας επανάληψης πάνω στην οποία είναι φτιαγμένο το παίγνιο.

Στόχος είναι να μπορούν να εφαρμοστούν αυτές οι 2 συναρτήσεις 60 φορές το δευτερόλεπτο και να παράγουν δηλαδή 60 frames per second (fps, καρέ ανά δευτερόλεπτο). Το XNA προσφέρει δύο διαφορετικούς τρόπους διαχείρισης του χρόνου του παιχνιδιού, δηλαδή δύο διαφορετικούς τρόπους συμπεριφοράς της κλίσης αυτών των συναρτήσεων.

Time Step

Fixed Time Step

Με αυτήν την τεχνική το XNA αμέσως μετά την ολοκλήρωση μιας Update και μιας Draw θα περιμένει όσο χρόνο χρειάζεται με σκοπό να υπάρχει σταθερός ρυθμός ανανέωσης (60 fps). Αν δεν υπήρχε αυτή η παύση ανάμεσα στην επανάληψη, ο ηλεκτρονικός υπολογιστής θα εκτελούσε το πρόγραμμα όσο το δυνατόν γρηγορότερα, με αποτέλεσμα να μην είναι ίδια η συμπεριφορά του παιχνιδιού σε διαφορετικούς υπολογιστές με διαφορετικές ικανότητες και φυσικά να μην υπάρχει σχετική ένια του χρόνου στο παιχνίδι.



Με ανώτατο όριο τα 60 fps μια επανάληψη θα πρέπει να επαναλαμβάνεται σε λιγότερο από 0.016 δευτερόλεπτα (με τον χρόνο που υπολείπεται να είναι η παύση). Στην περίπτωση όμως όπου ο χρόνος για την εκτέλεση της επανάληψης είναι μεγαλύτερος, οι μέθοδοι δεν θα καλούνται όσο συχνά χρειάζεται με αποτέλεσμα ο χρήστης να αντιλαμβάνεται την καθυστέρηση αυτήν.

Variable Time Step

Με αυτήν την τεχνική, παρακάμπτεται εντελώς το στάδιο της αναμονής και η μέθοδος Update εκτελείτε αμέσως μετά το τέλος της Draw. Σε αυτήν την περίπτωση ο προγραμματιστής θα πρέπει να φροντίσει να διατηρεί τον έλεγχο της ροής του χρόνου κατά την διάρκεια της εκτέλεσης του προγράμματος, μιας και δεν είναι πια πάντα σταθερός ο χρόνος από την μία επανάληψη στην άλλη και από έναν ηλεκτρονικό υπολογιστή σε έναν άλλο.

Στην περίπτωση όπου παρατηρηθεί μεγάλη καθυστέρηση ανάμεσα στις επαναλήψεις, το XNA αυτομάτως παρακάμπτει την Draw και επαναλαμβάνει την Update με σκοπό να ελαττώσει στον τον χρόνο εκτέλεσης. Μιας και η Update είναι υπεύθυνη για όλη την λογική του παιχνιδιού (αναλυτική περιγραφή όλων των συναρτήσεων παρακάτω), είναι αναγκαία η ολοκλήρωση της.

Το Variable Time Step αν και δυσκολότερο στην εφαρμογή του, είναι προτιμότερο του Fixed Time Step.

Initialize

Αυτή η μέθοδος είναι η πρώτη που εκτελείτε από τις πέντε βασικές μεθόδους, αμέσως μετά από τον constructor του παιχνιδιού. Σκοπός της είναι να δώσει έναν τρόπο στην προγραμματιστή να πραγματοποιήσει τυχών αρχικοποιήσεις (όπως τιμές μεταβλητών), πριν την εκτέλεση της λογικής του προγράμματος. Μπορεί να παραμείνει κενή.

LoadContent

Η LoadContent καλείτε αμέσως μετά την Initialize και σκοπός της είναι η φόρτωση των περιεχομένων του παιχνιδιού όπως 3D μοντέλα, textures, εικόνες, αρχεία ήχου, fonts και effects στην μνήμη.

Όπως είναι λογικό για την δημιουργία ενός παιχνιδιού δεν αρκούν τα περιεχόμενα μιας αντικειμενοστραφής γλώσσας προγραμματισμού. Χρειάζονται περισσότερα δεδομένα σε μορφές αρχείων όπως μια εικόνα, τα οποία θα πρέπει να φορτωθούν στην μνήμη από το πρόγραμμα. Τον σκοπό αυτό πραγματοποιεί αυτή η συνάρτηση, υποστηρίζοντας πολλούς γνωστούς τύπους αρχείων, αρκετούς ώστε να καλύψουν κάθε ανάγκη κατά την κατασκευή του παιχνιδιού.

Update

Η Update καλείτε μετά την LoadContent και είναι υπεύθυνη για ένα μεγάλο μέρος του προγράμματος, την λογική του. Με τον όρο λογική αναφερόμαστε σε οποιοδήποτε στοιχείο που κάνει το παιχνίδι αυτό που είναι, όπως οι κανόνες του, η φυσική του και η συμπεριφορά όλων των περιεχομένων του. Οτιδήποτε αφορά την πορεία του παιχνιδιού, ανά πάσα στιγμή, εκτελείτε μέσα στην Update.

Η Update είναι η πρώτη από τις δύο μεθόδους που επαναλαμβάνονται κατά την διάρκεια της εκτέλεσης του προγράμματος όπου γίνετε επεξεργασία όλων των δεδομένων που χρειάζονται ανά πάσα στιγμή για την λογική του παιχνιδιού όπως η θέση ενός αντικειμένου. Με την επανάληψη αυτής της μεθόδου έχουμε την συνεχής πορεία του παιχνιδιού.

Η Update παίρνει σαν όρισμα μια μεταβλητή τύπου GameTime η οποία αναφέρετε στον χρόνο που πέρασε από την προηγούμενη εκτέλεση της. Ο σκοπός αυτής της μεταβλητής είναι να μπορέσει ο προγραμματιστείς να έχει ένα σταθερό και βάσιμο τρόπο να ελέγχει την ροή του χρόνου στο πρόγραμμα. Για παράδειγμα αν στο παιχνίδι ο προγραμματιστής θέλει να εμφανιστεί ένα αντικείμενο τρία δευτερόλεπτα μετά από το πάτημα ενός κουμπιού, μπορεί να το κάνει δημιουργώντας μια μεταβλητή t=3 και κάθε φορά που καλείτε η Update να αφαιρεί το GameTime από αυτήν την μεταβλητή. Έτσι όταν t=0, τότε το αντικείμενο

μπορεί να εμφανιστεί, ανεξάρτητα από το πόσο γρήγορα εκτελείτε το παιχνίδι στον εκάστοτε Η/Υ.

Draw

Η Draw καλείτε μετά την Update και ο στόχος της είναι η εκτύπωση στην οθόνη των δεδομένων που χρειάζονται εκείνη την χρονική στιγμή στο παιχνίδι σύμφωνα με την λογική που πραγματοποιήθηκε στην Update. Όπως αναφέρθηκε πριν κατά την εκτέλεση του προγράμματος είναι πιθανό η Update και η Draw να μην προλαβαίνουν να ολοκληρωθούν στον χρόνο που θα έπρεπε. Σε αυτήν την περίπτωση παρακάμπτεται η Draw εντελώς δίνοντας στην Update χρόνο να προχωρήσει την λογική του παιχνιδιού πριν την επόμενη εκτύπωση στην οθόνη.

Σε κάθε επανάληψη της μεθόδου γίνεται καθαρισμός της οθόνης από την προηγούμενη εκτύπωση, επιλογή Render Target, εκτέλεση αρχείων με κώδικα για οπτικά effects γραμμένο σε HLSL και εκτύπωση με την χρήση της κλάσης SpriteBatch (για 2D).

Η Draw παίρνει σαν όρισμα μεταβλητή τύπου GameTime όπως η Update για τους ίδιους λόγους αν και σπανιότερη η ανάγκη της.

UnloadContent

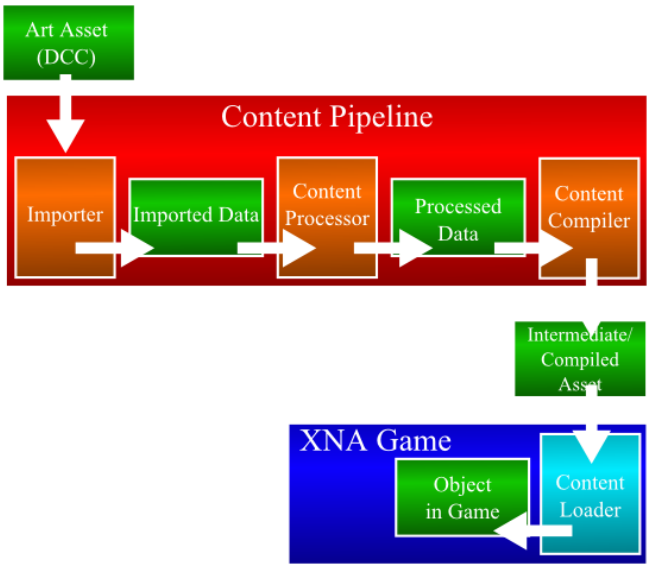
Αυτή η μέθοδος καλείτε στο τέλος του προγράμματος αλλά μπορεί και να κλιθεί όποτε επιθυμεί ο προγραμματιστείς. Σκοπός της είναι να δώσει την δυνατότητα να διαγραφεί από την μνήμη κάτι που έχει φορτωθεί από την LoadContent. Εφόσον με τον όρο contents αναφερόμαστε σε δεδομένα που δεν ανήκουν σε αυτά που διαχειρίζεται η γλώσσα προγραμματισμού (π.χ. με τον garbage collector) και έχουν φορτωθεί από τον προγραμματιστή χειροκίνητα,

είναι ανάγκη να υπάρχει ένας τρόπος να μπορούν να αφαιρεθούν από την μνήμη όταν πια δεν είναι χρήσιμα.

Λόγο της πολύ καλής διαχείρισης των contents από τον Content Manager, η ανάγκη για την κλίση της UnloadContent είναι σπάνια.

Content Pipeline

Το Content Pipeline είναι ένα πολύ χρήσιμο και δυνατό εργαλείο του XNA. Κάνει την διαχείριση των contents του παιχνιδιού πολύ ευκολότερη και βοηθάει στην διαφοροποίηση τους από το υπόλοιπο πρόγραμμα. Με το Content Pipeline κατά την δήλωση ενός αρχείου (π.χ. εικόνα ή ήχος) ως content, το XNA δημιουργεί ένα νέο αρχείο με τις πληροφορίες αυτού που του δόθηκε και όλες οι περεταίρω ενέργειες γίνονται πάνω σε αυτό. Με αυτόν τον τρόπο είναι επίσης εύκολο να διαχωριστεί η δουλειά του προγραμματιστή και η δουλειά του σχεδιαστή.



Τα βήματα του Content Pipeline είναι τα παρακάτω:

1. **Importer**. Σε αυτό το στάδιο το Content Pipeline διαβάζει το αρχείο που του δόθηκε.

2. **Content Processor**. Ο Content Processor επεξεργάζεται το αρχείο μελετώντας όλες τις πληροφορίες του όπως τον τύπο του περιεχομένου του.
3. **Content Compiler**. Ο Content Compiler παίρνει τα αποτελέσματα του Content Processor και δημιουργεί ένα νέο αρχείο έτοιμο για την χρήση του από το πρόγραμμα.
4. **Content Loader**. Ο Content Loader είναι υπεύθυνος για την ανάγνωση και φόρτωση του αρχείου στην μνήμη, όταν αυτό ζητηθεί από τον προγραμματιστή.

Content Manager

Ο Content Manager είναι μια κλάση του XNA Framework φτιαγμένη για δώσει τα απαραίτητα εργαλεία στον προγραμματιστή για την διαχείριση των contents. Κατά την δημιουργία ενός XNA project παράγεται αυτομάτως ένα instance του Content Manager, με την χρήση του οποίου μπορεί να γίνει το Load και Unload των contents.

Είναι σημαντικό να αναφερθεί ότι το κάθε instance του Content Manager διαχειρίζεται την μνήμη που καταλαμβάνουν τα contents που φορτώθηκαν με αυτό. Με την καταστροφή του instance ελευθερώνεται αυτομάτως η μνήμη που καταλαμβάνουν τα contents του, χωρίς να χρειάζεται δηλαδή να το κάνει χειροκίνητα ο προγραμματιστής. Έτσι, με την δημιουργία ξεχωριστών instances του Content Manager και την κατανομή των contents σε αυτά (π.χ. ανά στάδιο του παιχνιδιού), μπορεί να γίνει εύκολα σωστή διαχείριση της μνήμης.

SpriteBatch

Η SpriteBatch είναι μια από τις σημαντικότερες κλάσεις του XNA για την δημιουργία 2D παιχνιδιών. Ένα αντικείμενο αυτής της κλάσεις έχει όλα τα εργαλεία που χρειάζονται για την εκτύπωση 2D γραφικών στην οθόνη. Έστω πως δημιουργούμε ένα αντικείμενο αυτής της κλάσεις με το όνομα thespritebatch.

- Με την εντολή thespritebatch.**Begin()** ξεκινάμε την διαδικασία της εκτύπωσης. Αυτή η μέθοδος έχει πέντε overloads και επτά πιθανά ορίσματα. Τα σημαντικότερα είναι:

- **SpriteSortMode** όπου επιλέγουμε πότε και με τι σειρά θα γίνει η εκτύπωση
- **BlendState** όπου επιλέγουμε το είδος μίξης των χρωμάτων ανά pixel
- **Effect** όπου επιλέγουμε το αρχείο με των κώδικα των effects που θέλουμε να χρησιμοποιήσουμε σε αυτήν την εκτύπωση
- **Matrix** όπου δίνουμε στην συνάρτηση της εκτύπωσης το matrix που επιθυμούμε

- Με την εντολή thespritebatch.**Draw()** εκτυπώνουμε στην οθόνη το sprite που θέλουμε με τις επιλογές που επιθυμούμε. Η εντολή Draw έχει επτά overloads και εννιά ορίσματα τα οποία είναι:
  - Texture2D: το αρχείο της εικόνας.
  - Vector2: η θέση του αντικειμένου σε μορφή διανύσματος.
  - Rectangle: ένα ορθογώνιο που περιβάλει το κομμάτι της αρχικής εικόνας που θέλουμε να προβάλουμε.
  - Color: το χρώμα της εικόνας. Επιλέγοντας άσπρο παραμένει ως έχει.
  - Singe: η γωνία του αντικειμένου.
  - Vector2: το σημείο αναφοράς του αντικειμένου.
  - Vector2: η κλίμακα του αντικειμένου σε μορφή διανύσματος.
  - SpriteEffect: εδώ μπορούμε να ορίσουμε ένα από τα effects που υπάρχουν είδη στο XNA για sprites όπως πχ η περιστροφή του.
  - Single: το βάθος.

- Με την εντολή thespritebatch.**End**() τερματίζεται η διαδικασία της εκτύπωσης.

Shaders

Στον τομέα των γραφικών του υπολογιστή , ένα shader είναι ένα πρόγραμμα υπολογιστή που χρησιμοποιείται για να κάνει σκίαση : η παραγωγή των κατάλληλων επιπέδων του χρώματος μέσα σε μια εικόνα, ή επίσης να παράγουν ειδικά εφέ. Είναι δηλαδή ένα πρόγραμμα που λέει στον Η/Υ πώς να ζωγραφίσει κάτι, με έναν ξεχωριστό και μοναδικό τρόπο.

Οι Shaders υλοποιούν rendering effects με την χρήση καρτών γραφικών. Οι περισσότεροι shaders κωδικοποιούνται για μονάδες επεξεργασίας γραφικών (GPU), αν και αυτό δεν είναι απαραίτητο. Οι γλώσσες σκίασης χρησιμοποιείται για να προγραμματίσουν το GPU rendering pipeline , το οποίο αντικατέστησε κυρίως τον αγωγό σταθερής λειτουργίας που επιτρέπει μόνο γεωμετρικούς μετασχηματισμούς και pixel shading λειτουργίες. Η θέση, απόχρωση, κορεσμός, φωτεινότητα και η αντίθεση όλων των pixel , vertices , ή textures που χρησιμοποιούνται για την κατασκευή της τελική εικόνα μπορούν να τροποποιηθούν ανά πάσα στιγμή, χρησιμοποιώντας αλγόριθμους που ορίζονται στο shader όπως επίσης και με εξωτερικές μεταβλητές ή textures που ορίζονται κατά την κλήση του shader.

2D Animations και Texture Atlases

Στον κόσμο των 2D εφαρμογών, είναι αρκετά απλό να παρουσιάσει κανείς διάφορα sprites οπουδήποτε θέλει στην οθόνη ή να τα κάνει να κινούνται όπως για παράδειγμα ένας χαρακτήρας που περπατάει. Δεν αρκεί όμως αυτό γιατί πέρα από την μετατόπιση του στον χώρο, πρέπει και ο ίδιος ο χαρακτήρας να δείχνει ότι κινείτε, να έχει δηλαδή ένα animation.

Τα 2D animations δεν είναι τίποτα άλλο από διάφορες εικόνες που εναλλάσσονται γρήγορα με αποτέλεσμα την αίσθηση της κίνησης. Όπως και σε ένα film μιας ταινίας, τα animations αποτελούνται από

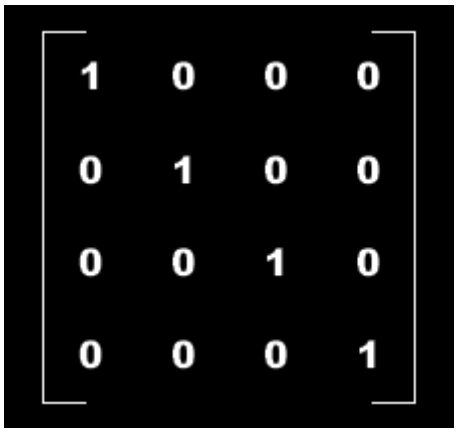
στιγμιότυπα της κίνησης που θέλουμε να περιγράψουμε και παρουσιάζονται διαδοχικά στην οθόνη.

Αντί όμως για την χρήση πολλών διαφορετικών αρχείων εικόνας και πολλών sprites, για την περιγραφή ενός animation είναι προτιμότερη η χρήση ενός Texture Atlas. Το Texture Atlas είναι μια εικόνα που περιέχει όλα τα καρέ της κίνησης. Την δυνατότητα να παράγουμε το animation με αυτόν τον τρόπο μας δίνει το τρίτο όρισμα της συνάρτησης draw του spritebatch. Με την χρήση αυτού του τετραγώνου μπορούμε να επιλέξουμε το κομμάτι της αρχικής εικόνας που θέλουμε να εμφανιστεί. Έτσι για την παραγωγή ενός animation χρησιμοποιούμε μόνο ένα αρχείο, από το οποίο εμφανίζουμε κάθε φορά διαφορετικό του κομμάτι.

Matrices και Γραμμική Άλγεβρα

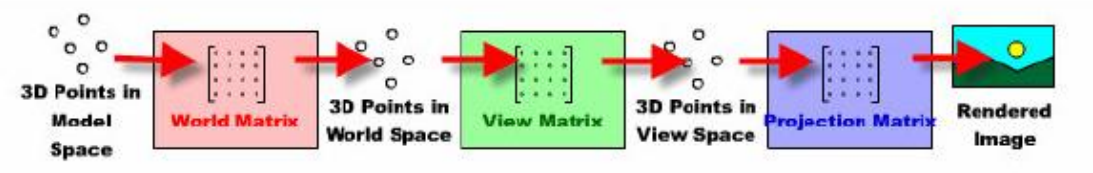
Η γραμμική άλγεβρα είναι τομέας των μαθηματικών και της άλγεβρας ο οποίος ασχολείται με τη μελέτη διανυσμάτων, διανυσματικών χώρων, γραμμικών απεικονίσεων και συστημάτων γραμμικών εξισώσεων. Η αναλυτική γεωμετρία αποτελεί έκφρασή της και η ίδια αποτελεί κεντρικό συνδετικό ιστό των σύγχρονων μαθηματικών, ιδιαιτέρως μέσω της αφηρημένης έννοιας του διανυσματικού χώρου η οποία μπορεί να μοντελοποιήσει πολλά διαφορετικά προβλήματα που συναντώνται στην πράξη.

Συνηθισμένη πρακτική είναι η προσέγγιση μη γραμμικών φαινομένων με γραμμικά μοντέλα (γραμμικοποίηση), προκειμένου να μπορούν να εφαρμοστούν οι μεθοδολογίες της γραμμικής άλγεβρας. Η εν λόγω «γραμμικότητα» αφορά το γεγονός ότι οι μεθοδολογίες αυτές εφαρμόζονται σε σύνολα συναρτήσεων οι οποίες στον τύπο τους περιέχουν μόνο πολυώνυμα πρώτου ή μηδενικού βαθμού και περιγράφουν σχέσεις μεταξύ ν-διάστατων διανυσμάτων. Οι συναρτήσεις αυτές ονομάζονται και γραμμικές επειδή, στην αναλυτική γεωμετρία, απεικονίζονται οπτικά με ευθείες γραμμές.



Στην παραπάνω εικόνα απεικονίζετε ένα matrix. Ένα matrix μπορεί να έχει οποιοδήποτε μέγεθος και να περιέχει διάφορες τιμές (όχι μόνο 0 ή 1 όπως στην εικόνα). Η χρήση τους στα ηλεκτρονικά παιχνίδια είναι η μετατροπή συντεταγμένων από ένα σύστημα σε άλλο, χωρίς απώλειες. Ένα παιχνίδι περιέχει συνήθως τρία βασικά matrices.

- **World Matrix**: Έστω ότι έχουμε ένα μοντέλο. Οι συντεταγμένες των διανυσμάτων μας περιγράφουν την θέση τους σε σχέση με το μοντέλο. Με την χρήση του world matrix μπορούμε να τις μετατρέψουμε από model-space σε world-space. Με αυτόν τον τρόπο οι συντεταγμένες τώρα μας υποδηλώνουν την θέση του αντικειμένου σε σχέση με τον κόσμο του παιχνιδιού.
- **View Matrix**: Χρησιμοποιώντας τις συντεταγμένες μας από το προηγούμενο βήμα (world-space) και μετατρέποντας τες με το view matrix παίρνουμε τις συντεταγμένες του αντικειμένου σε σχέση με τον θεατή, δηλαδή την θέση της κάμερας.
- **Projection Matrix**: Μετατρέποντας τα αποτελέσματα του view matrix με το projection matrix ουσιαστικά παίρνουμε σαν αποτέλεσμα τον «τύπο» κάμερας του παιχνιδιού. Δίνονται στην υπολογιστή οι απαραίτητες πληροφορίες έτσι ώστε να γνωρίζει που βρίσκετε το μοντέλο μας στην οθόνη και έτσι έχουμε τις αντίστοιχες συντεταγμένες (screen space).



Περιγραφή του Παιγνίου

Στόχος της Δημιουργίας

Πρόκειται για ένα διδιάστατο arcade-puzzle παιχνίδι, φτιαγμένο στο MonoGame με την χρήση του Microsoft XNA Framework. Σκοπός της κατασκευής του είναι η έρευνα των τεχνολογιών που υπάρχουν στον χώρο αυτό, το επίπεδο πολυπλοκότητας και δυσκολίας ανάπτυξης στην χρήση τους και η προσβασιμότητα τους. Ένα ακόμα στόχος της δημιουργίας του είναι η μελέτη των δυνατοτήτων του, τόσο στον χώρο των computer games, όσο και για άλλους σκοπούς όπως εκπαιδευτικούς ή ερευνητικούς.

Τα αποτελέσματα της έρευνας αυτής είναι κατά κύριο λόγο θετικά. Οι τεχνολογίες στον χώρο των ηλεκτρονικών παιχνιδιών βρίσκονται σε υψηλό επίπεδο και εξελίσσονται καθημερινά από τα μεγαλύτερα ονόματα στον τομέα της τεχνολογίας. Η πρόοδος της τεχνολογίας των παιχνιδιών μπορεί να παρατηρηθεί άμεσα ακόμα και από κάποιον χωρίς εμπειρία σε αυτόν τον τομέα. Με την εξέλιξη όμως της τεχνολογίας και τις δυνατότητες που προσφέρει είναι λογικό πως και η πολυπλοκότητα κατασκευής αυξάνεται. Με τεχνικές όμως όπως η χρήση μηχανών παιχνιδιού, το επίπεδο της δυσκολίας δεν είναι ανάλογο της συνολικής πολυπλοκότητας για την δημιουργία ενός παιχνιδιού. Αυτές οι τεχνικές όμως παραμένουν στα χέρια των εταιριών που τις ανέπτυξαν με αποτέλεσμα να είναι δύσκολο για μια εταιρία ή ομάδα ατόμων να κατασκευάσει ένα ανταγωνιστικό παιχνίδι.

Με την χρήση του MonoGame μπορεί κανείς να δημιουργήσει ένα παιχνίδι και να το προωθήσει σχεδόν σε όλες τις πλατφόρμες. Όπως μπορεί να παρατηρήσει κανείς σήμερα, ο κόσμος των ηλεκτρονικών παιχνιδιών είναι γεμάτος με μικρές και λιγότερο σύνθετες εφαρμογές συγκριτικά με τα μεγάλα ονόματα του χώρου που όμως πολλά από αυτά έχουν γίνει με τον καιρό μεγάλοι τίτλοι προσφέροντας μεγάλα κέρδη στους κατασκευαστές τους. Το MonoGame δίνει την δυνατότητα

για κάτι τέτοιο και αν και δεν είναι ακόμα ολοκληρωμένο, υπάρχουν αρκετοί γνωστοί τίτλοι κατασκευασμένοι σε αυτό. Φυσικά εκτός από εμπορικούς λόγους, μια τέτοια εφαρμογή μπορεί να χρησιμοποιηθεί για έρευνα εκπαίδευση, από απλά παιχνίδια αριθμητικής για παιδία, μέχρι εξομοιώσεις φυσικών φαινομένων.

Γενικά

Το πρόγραμμα περιλαμβάνει δύο αρκετά διαφορετικά modes παιχνιδιού, ολοκληρωμένο μενού, ρυθμίσεις, οδηγίες και επεξήγηση για κάθε mode και οθόνη με τις μεγαλύτερες βαθμολογίες ανά mode.

Mode 1 (black hole)

Σε αυτό το mode ο παίκτης χειρίζεται μια μαύρη τρύπα στο διάστημα, ενώ περιβάλεται από φλεγόμενους μετεωρίτες. Κάθε ένας από αυτούς έχει έναν αριθμό από το μηδέν μέχρι το 6. Σκοπός του παιχνιδιού είναι κινώντας την μαύρη τρύπα να απορροφήσεις μετεωρίτες προσθέτοντας τον αριθμό τους στον δικό της και να σχηματίσεις δεκάδες. Κάθε φορά που ολοκληρώνεται μία δεκάδα ο αριθμός της μαύρης τρύπας μηδενίζει. Πραγματοποιώντας πρόσθεση η οποία έχει σαν αποτέλεσμα την υπέρβαση του δέκα σαν αριθμό για αυτήν οδηγεί στην λήξη του παιχνιδιού (game over). Όσο ο χρήστης καταστρέφει μετεωρίτες τόσο νέοι θα εμφανίζονται από τα άκρα της οθόνης με τυχαίους αριθμούς αλλά σταδιακά με όλο και μεγαλύτερη αρχική ταχύτητα. Το σκορ του χρήστη σε αυτό το mode είναι το γενικό άθροισμα που έχει απορροφήσει. Ένας μετεωρίτης με αριθμό μηδέν, προσθέτει αυτόματα μια δεκάδα στο συνολικό σκορ και δεν επηρεάζει τον αριθμό της μαύρης τρύπας.

Σε αυτό το mode επίσης εμφανίζονται τα ακόλουθα δύο powerups.

Όταν η μαύρη τρύπα συγκρουστεί με το πρώτο (μπλε) τότε η ταχύτητα των μετεωριτών μειώνεται για ένα χρονικό διάστημα, ενώ με το δεύτερο (πορτοκαλί) αυξάνεται.

Το mode αυτό παίζετε μόνο από έναν παίκτη, χρησιμοποιώντας το ποντίκι, το πληκτρολόγιο ή ακόμα και κάποιο joypad.

Mode 2 (υποβρύχιο)

Σε αυτό το mode ο παίκτης χειρίζεται ένα υποβρύχιο το οποίο έχει τρία βασικά χαρακτηριστικά, ζωή, ασπίδα και πυρομαχικά (Health, Armor, Ammo). Ανά τακτά χρονικά διαστήματα, νάρκες εισέρχονται από το πάνω μέρος της οθόνης και βυθίζονται στο νερό με μεγάλη ταχύτητα και έπειτα κινούνται αργά προς την επιφάνεια μέχρι να φτάσουν στο βάθος στο οποίο και θα ισορροπήσουν μέσα στο νερό. Σκοπός του παιχνιδιού είναι η αποφυγή των ναρκών και παράλληλα η καταστροφή τους με την χρήση πυρομαχικών.

Κατά την σύγκρουση τους με το υποβρύχιο ή με μια τορπίλη του, οι νάρκες καταστρέφονται ακαριαία. Άλλες νάρκες που τυχών βρίσκονται σε κοντινή απόσταση με αυτήν που μόλις εξεράγει, θα ενεργοποιήσουν το σύστημα αυτοκαταστροφής τους με αποτέλεσμα να εκραγούν και αυτές μετά από λίγα δευτερόλεπτα. Το ίδιο μπορεί να συμβεί και σε περίπτωση σύγκρουσης δύο ναρκών, κάτι που εξαρτάτε από την δύναμη της σύγκρουσης.

Με την πρόσκρουση μιας νάρκης στο υποβρύχιο ή την έκρηξη της κοντά σε αυτό, το υποβρύχιο αυτομάτως θα χρησιμοποιήσει τις ασπίδες του αν υπάρχουν για την αποφυγή ζημιάς, αναλώνοντας τες. Στην περίπτωση που δεν υπάρχουν ασπίδες το υποβρύχιο δέχεται ζημιά, δηλαδή μείωση της ζωής του.

Όταν ο χρήσης καταστρέψει μια νάρκη με την χρήση πυρομαχικών, δηλαδή τορπιλών, επιβραβεύεται με πόντους ανάλογους της απόστασης της νάρκης από το υποβρύχιο και της ταχύτητας της την ώρα της σύγκρουσης. Επίσης με την συλλογή πόντων το υποβρύχιο

αναπληρώνει έναν μικρό αριθμό τορπιλών. Υπάρχουν δύο είδη τορπιλών, οι απλές και οι ειδικές.

Σε αυτό το mode υπάρχουν τρία διαφορετικά powerups. Το ένα αναπληρώνει ένα μέρος της ζωής του υποβρυχίου, το δεύτερο τορπίλες και το τρίτο δίνει τρεις ασπίδες στο υποβρύχιο.

Αυτό το mode μπορεί ναι παιχτεί με έναν ή και με δύο παίκτες. Στην περίπτωση των δύο παικτών ο στόχος είναι πάλι το μεγαλύτερο σκορ. Όταν κάποιος παίκτης εξοντωθεί, το παιχνίδι συνεχίζεται με αυτόν που απομένει. Όσο υπάρχουν δύο παίκτες ενεργοί οι ειδικές τορπίλες ταξιδεύουν αυτόματα προς τον αντίπαλο παίκτη με σκοπό να συγκρουστούν πάνω του, ενώ στην περίπτωση του ενός παίκτη η ειδική τορπίλη θα ψάξει και θα προσπαθήσει να συγκρουστεί με την νάρκη που θα αποδώσει τους περισσότερους πόντους. Το mode αυτό μπορεί να παιχτεί με την χρήση του πληκτρολογίου ή joypad.

Μενού

Το μενού του παιχνιδιού βρίσκετε στο αριστερό μέρος της οθόνης και περιλαμβάνει τις παρακάτω επιλογές:

1. **NEW GAME**: Επιλέγοντας New Game ξεκινάει το παιχνίδι, σύμφωνα με το επιλεγμένο mode.
2. **OPTIONS**: Τα Options είναι ένα σημαντικό κομμάτι του προγράμματος καθώς περιλαμβάνουν όλες τις ρυθμίσεις που το αφορούν. Ακολουθεί η λίστα αυτών των ρυθμίσεων.

    a. Window Resolution: Εδώ επιλέγεται το επιθυμητό μέγεθος του παραθύρου.

b. Fullscreen Mode: Εδώ ενεργοποιείται η επιλογή της πλήρης οθόνης.
c. Music Volume level: Εδώ ρυθμίζεται η ένταση της μουσικής του παιχνιδιού.
d. Sound FX Volume level: Εδώ ρυθμίζεται η ένταση των ηχητικών εφέ.
e. Modes: Από επιλέγεται το mode που θέλουμε να παίξουμε.
f. Themes: Εδώ υπάρχουν τα θέματα για κάθε mode.
g. Particles level: Εδώ επιλέγεται το επίπεδο ακρίβειας του particle engine.
h. Apply: Από εδώ επιβεβαιώνονται και αποθηκεύονται όλες οι ρυθμίσεις.
i. Main Menu: Πατώντας εδώ γίνεται επιστροφή στο αρχικό μενού.

3. **SCOREBOARD**: Σε αυτήν την οθόνη μπορεί κανείς να δει τα μεγαλύτερα σκορ που έχουν γίνει στο mode που είναι επιλεγμένο εκείνη την στιγμή.
4. **ABOUT**: Σε αυτήν την οθόνη μαθαίνουμε πληροφορίες για το κάθε mode.
5. **EXIT**: Πατώντας εδώ κλείνει το πρόγραμμα.

Τεχνικά Χαρακτηριστικά του Παιγνίου

Για την δημιουργία του παιχνιδιού ήταν αναγκαία η ανάπτυξη και η χρήση διαφόρων τεχνικών και τεχνολογιών. Σε αυτήν την παράγραφο γίνετε αναφορά και επεξήγηση των σημαντικότερων και παρουσίαση κομματιών κώδικα.

Οθόνες

Όπως αναφέρθηκε πριν, το παιχνίδι περιλαμβάνει διάφορες οθόνες όπως η οθόνη των OPTIONS ή του ABOUT. Υπάρχουν διάφοροι τρόποι για την δημιουργία οθονών όπως η χρήση ENUMERATIONS. Για την συγκεκριμένη εφαρμογή επέλεξα την χρήση μιας ιδιότητας του αντικειμενοστραφούς προγραμματισμού, τον **πολυμορφισμό**. Έτσι, με

την χρήση μιας κεντρικής κλάσης screen, η οποία περιέχει τις βασικές συναρτήσεις και ιδιότητες μιας οθόνης, δημιούργησα μια κλάση για κάθε οθόνη, που όλες τους κληρονομούν την κλάση screen. Με αυτόν τον τρόπο έχω την δυνατότητα να ορίσω μια μεταβλητή τύπου screen στο πρόγραμμα μου, την currentscreen, η οποία χρησιμοποιείται σαν δείκτης της τρέχουσας οθόνης στο παιχνίδι. Έτσι μπορώ να ορίσω σαν currentscreen οποιοδήποτε παιδί από τις κλάσεις που αντιπροσωπεύουν τις οθόνες αλλά παράλληλα κάθε φορά το currentscreen να έχει διαφορετικό περιεχόμενο.

```
public class mScreen
    {
        protected EventHandler ScreenEvent;
        public mScreen(EventHandler theScreenEvent)
        {
            Game1.Loading = true;
            ScreenEvent = theScreenEvent;
        }

        public virtual void LoadContent(ContentManager theContentManager)
        {
            Game1.Loading = false;
        }
        public virtual void Update(GameTime theTime)
        {
        }
        public virtual void Draw(SpriteBatch theBatch)
        {
        }
    }

class AboutScreen : mScreen
    {
       …
        public AboutScreen(EventHandler theScreenEvent, bool isfrompause)
            : base(theScreenEvent)
        {
            pause = isfrompause;
        }
public override void LoadContent(Microsoft.Xna.Framework.Content.ContentManager
theContentManager)
        {
            …
            base.LoadContent(theContentManager);
        }
public override void Update(Microsoft.Xna.Framework.GameTime theTime)
        {
            …
            base.Update(theTime);
        }
public override void Draw(Microsoft.Xna.Framework.Graphics.SpriteBatch
theBatch)
        {
            …
            base.Draw(theBatch);
```

```
        }
    }
```

Σε αυτό το παράδειγμα έχουμε την κλάση AboutScreen η οποία κληρονομεί την mScreen. Το παιδί της AboutScreen είναι η οθόνη ABOUT. Όταν επιλέγουμε από το μενού του παιχνιδιού αυτήν την οθόνη, τότε η currentscreen γίνετε ίση με το παιδί αυτής της κλάσης και εκτελούνται αντίστοιχα οι παραπάνω συναρτήσεις (τα περιεχόμενα τους είναι κενά λόγο μεγέθους). Το ίδιο συμβαίνει και για τις άλλες οθόνες του παιχνιδιού.

Φυσική

Σε ένα παιχνίδι σαν αυτό είναι έντονη η χρήση νόμων της φυσικής για την περιγραφή και την συμπεριφορά του παιχνιδιού. Τα βασικότερα κομμάτια είναι η κίνηση στον χώρο και η ελαστική κρούση.

Κίνηση (Motion)

Όπως αναφέρθηκε σε προηγούμενο κεφάλαιο, κάθε παιχνίδι αποτελεί μια σειρά από στιγμιότυπα. Κίνηση είναι η μετατόπιση του αντικειμένου στην πάροδο του χρόνου δηλαδή η αλλαγή της θέσης του σε κάθε καρέ. Αυτό επιτυγχάνεται με την χρήση του διανύσματος της θέσης του αντικειμένου, του διανύσματος της ταχύτητας και της χρονικής αναφοράς της πορείας του παιχνιδιού (GameTime).

```
public void Move(GameTime gameTime, Mode1Object movingobject)
        {
            movingobject.Position += movingobject.Velocity *
(float)gameTime.ElapsedGameTime.TotalSeconds;
                                        }
```

Στην συνάρτηση αυτή δίνεται το GameTime και το αντικείμενο που θέλουμε να μετακινήσουμε. Επειδή το πρόγραμμα περιέχει διαφόρων ειδών κλάσεις που περιγράφουν τα αντικείμενα, έγινε χρήση του πολυμορφισμού όπως αναφέρθηκε πριν. Η μεταβλητή Position αντιπροσωπεύει το διάνυσμα θέσης και η Velocity το διάνυσμα ταχύτητας. Την νέα θέση του αντικειμένου την παίρνουμε από την

πρόσθεση της τρέχουσας θέσης και της ταχύτητας σε σχέση με τον χρόνο που πέρασε από την προηγούμενη κλήση της Update.

Ελαστική Κρούση

Κατά την ελαστική κρούση δύο σωμάτων, διατηρείται και η ορμή και η κινητική ενέργεια των σωμάτων. Επομένως ισχύουν οι παρακάτω σχέσεις:

Αρχή διατήρησης της ορμής:

$$\mathbf{m}_1 u_1 + m_2 u_2 = m_1 v_1 + m_2 v_2.$$

Αρχή διατήρησης της ενέργειας:

$$\frac{m_1 u_1^2}{2} + \frac{m_2 u_2^2}{2} = \frac{m_1 v_1^2}{2} + \frac{m_2 v_2^2}{2}.$$

(όπου **u** οι ταχύτητες των σωμάτων πριν την κρούση και **υ** η ταχύτητα των σωμάτων μετά την κρούση)

Από την επίλυση του συστήματος των παραπάνω εξισώσεων προκύπτει ότι η ταχύτητα των σωμάτων μετά την κρούση δίνεται από τις σχέσεις:

$$v_1 = \frac{u_1(m_1 - m_2) + 2m_2 u_2}{m_1 + m_2}, \quad v_2 = \frac{u_2(m_2 - m_1) + 2m_1 u_1}{m_1 + m_2}$$

Για την εξομοίωση της ελαστικής κρούσης με την χρήση διανυσμάτων χρησιμοποιήθηκε η μέθοδος από αυτό το αρχείο http://www.imada.sdu.dk/~rolf/Edu/DM815/E10/2dcollisions.pdf.

```
public void bounce(Mode1Object o1, Mode1Object o2)
        {
            Vector2 un = new Vector2(o2.Position.X - o1.Position.X,
o2.Position.Y - o1.Position.Y);
            un.Normalize();
            Vector2 ut = new Vector2(-un.Y, un.X);
            float v1n = Vector2.Dot(un, o1.Velocity);
            float v1t = Vector2.Dot(ut, o1.Velocity);
            float v2n = Vector2.Dot(un, o2.Velocity);
            float v2t = Vector2.Dot(ut, o2.Velocity);
            float v1nnew = (v1n * (o1.mass - o2.mass) + (2 * o2.mass * v2n)) /
(o1.mass + o2.mass);
```

```
            float v2nnew = (v2n * (o2.mass - o1.mass) + (2 * o1.mass * v1n)) /
(o2.mass + o1.mass);
            Vector2 v1n_vector = v1nnew * un;
            Vector2 v1t_vector = v1t * ut;
            Vector2 v2n_vector = v2nnew * un;
            Vector2 v2t_vector = v2t * ut;
            o1.Velocity = v1n_vector + v1t_vector;
            o2.Velocity = v2n_vector + v2t_vector;

                                        }
```

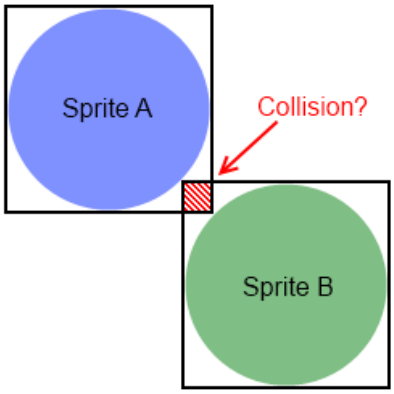
Αυτή η συνάρτηση δέχεται σαν ορίσματα τα δύο αντικείμενα της σύγκρουσης και υπολογίζει την νέα διανυσματική ταχύτητα τους.

Είναι σημαντικό να σημειώσουμε πως δεν αρκεί μόνο η ικανότητα της εύρεσης του αποτελέσματος της κρούσης, αλλά και η ανίχνευση της ίδιας της κρούσης κατά την διάρκεια της εκτέλεσης του προγράμματος.

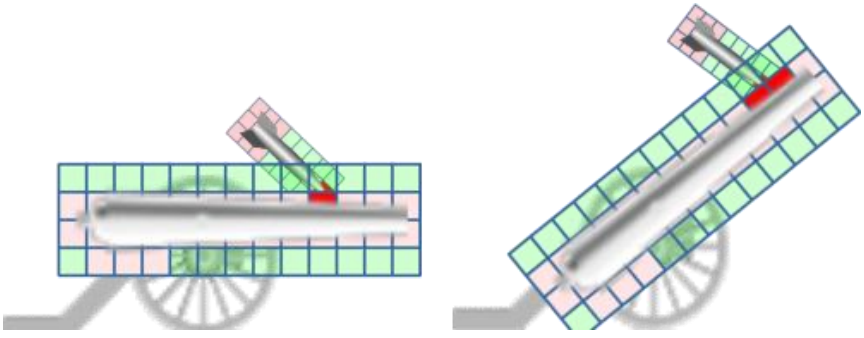
Το XNA είναι ικανό να διαχειριστεί βασικά σχήματα όπως ο κύκλος και το ορθογώνιο και να ανιχνεύσει τυχών κρούσεις αντικειμένων αυτών των σχημάτων. Αν και όλα τα πιθανώς συγκρουόμενα αντικείμενα της εφαρμογής έχουν κυκλικό σχήμα, προτίμησα για ερευνητικούς λόγους να αναπτύξω δικές μου μεθόδους για την εύρεση συγκρούσεων μετάξι αντικειμένων.

Όταν ένα αντικείμενο περιγράφετε από ένα τέτοιο βασικό σχήμα και η θέση αυτού του σχήματος (άρα και του αντικειμένου) δοθεί στο XNA, τότε αυτό μπορεί να μας πει αν δύο τέτοια σώματα συγκρούονται. Σε κάθε άλλη περίπτωση έχουμε δύο επιλογές.

Η πρώτη είναι να περιβάλουμε το αντικείμενο μας (ότι μορφή και αν έχει) σε ένα βασικό σχήμα όπως το ορθογώνιο παραλληλόγραμμο και να το επεξεργαστούμε θεωρώντας άκρα του τα άκρα του ορθογωνίου με τον παραπάνω τρόπο, αφήνοντας δηλαδή το XNA να αποφασίσει να υπάρχει σύγκρουση. Αυτή η μέθοδος είναι πολύ καλή από άποψη χρόνου εκτέλεσης της αλλά έχει το μειονέκτημα ότι εφόσον υπάρχει διαφορά ανάμεσα στο ορθογώνιο και το ίδιο το αντικείμενο, είναι πολύ πιθανό να ανιχνεύεται κρούση (δηλαδή επαφή των τετραγώνων που περιβάλουν τα αντικείμενα) χωρίς τα ίδια τα αντικείμενα να έχουν έρθει σε επαφή.



Η άλλη εναλλακτική μέθοδος μας εγγυάται μεγαλύτερη ακρίβεια αλλά και μεγαλύτερο χρόνο εκτέλεσης. Σε αυτήν την περίπτωση δημιουργούμε έναν πίνακα χρωμάτων για το κάθε αντικείμενο όπου κάθε στοιχείο του πίνακα περιέχει το χρώμα του pixel του αντικειμένου, για κάθε pixel του. Ένα χρώμα περιγράφετε από τέσσερις τιμές συνολικά, τρεις για το χρώμα του (R,G,B) και A για την αδιαφάνεια του (opacity). Στο πρόγραμμα μας, κάθε φορά που θέλουμε να μελετήσουμε δύο αντικείμενα για πιθανή κρούση, παίρνουμε αυτόν τον πίνακα και με την χρήση των matrices, συγκρίνουμε ένα προς ένα τα pixels τους για την περίπτωση όπου οι συντεταγμένες κάποιου pixel του ενός αντικειμένου στο world space είναι ίδιες με τις συντεταγμένες κάποιου pixel του άλλου αντικειμένου. Αυτό όμως δεν αρκεί γιατί θα μπορούσε κάποιο από αυτά τα δύο pixels να έχουν αδιαφάνεια μηδενική, όπως το pixel της προηγούμενης εικόνας. Έτσι πρέπει τώρα να συγκρίνουμε με την χρήση των πινάκων χρωμάτων τα Alpha των pixels αυτών. Αν και τα δύο έχουν Alpha διάφορο του μηδενός, τότε έχουμε σύγκρουση. Αυτή η μέθοδος ονομάζεται και pixel perfect collision method.



Σε γενικές γραμμές είναι καλύτερο να χρησιμοποιούμε την πρώτη μέθοδο αν η εφαρμογή μας δεν έχει ανάγκη για ακρίβεια όπως γίνετε σε πολλά γνωστά παιχνίδια. Στην περίπτωση όμως που αυτή η μέθοδος δεν καλύπτει τις ανάγκες μας, μπορούμε να δουλέψουμε με τον συνδυασμό των δύο μεθόδων. Αρχικά θα γίνετε μια πρώτη ανίχνευση σύγκρουσης με την πρώτη μέθοδο και αν υπάρχει σύγκρουση τότε μπορούμε να χρησιμοποιήσουμε την pixel perfect collision method για να αληθεύσουμε με ακρίβεια την ύπαρξη της.

Particle Engine

Ένα Particle Engine είναι μια τεχνική της φυσικής παιχνιδιών και των γραφικών υπολογιστών που χρησιμοποιεί ένα μεγάλο αριθμό πολύ μικρών sprites ή άλλων αντικείμενων γραφικών για την προσομοίωση ορισμένων φαινομένων, τα οποία είναι κατά τα άλλα πολύ δύσκολο να αναπαραχθούν με τις συμβατικές τεχνικές rendering - συνήθως ιδιαίτερα χαοτικά συστήματα, φυσικά φαινόμενα, ή διαδικασίες που προκαλούνται από χημικές αντιδράσεις.

Στην συγκεκριμένη εφαρμογή το particle engine που δημιούργησα έχει σαν σκοπό την διαχείριση των σωματιδίων φωτιάς που αφήνει πίσω του ένας μετεωρίτης καθώς κινείται (mode 1) και τις φυσαλίδες που παράγονται με την κίνηση του υποβρυχίου και των τορπιλών (mode 2).

Ας πάρουμε για παράδειγμα το πρώτο mode. Με την κίνηση των μετεωριτών, το particle engine παράγει αντικείμενα της κλάσεις particle όπου περιέχουν texture, διάνυσμα ταχύτητας, διάνυσμα θέσης, μέγεθος και χρόνο ζωής. Για texture έχει την δυνατότητα να επιλέξει τυχαία από μία λίστα μικρών εικόνων κύκλων με αποχρώσεις του πορτοκαλί και του κόκκινου. Του δίνει σαν θέση κάποιο σημείο στο πίσω μέρος του μετεωρίτη όπως αυτός κινείται και σαν ταχύτητα μια μικρή τυχαία τιμή, αντίθετη της ταχύτητας του μετεωρίτη. Ο χρόνος ζωής του κυμαίνεται από 0.1 έως 2 δευτερόλεπτα.

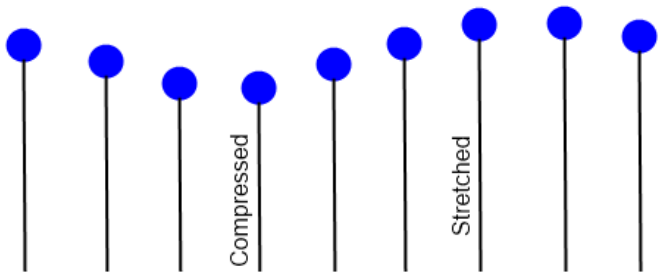
Σε κάθε καρέ κατά την εκτέλεση του προγράμματος το particle engine ελέγχει όλους τους μετεωρίτες και για τον καθένα φροντίζει να

διαγράψει τα particles που ο χρόνος ζωής τους έχει εξαντληθεί, να δημιουργήσει νέα μέχρι να φτάσει τον μέγιστο αριθμό και να μειώσει τον υπολειπόμενο χρόνο ζωής των υπαρχόντων particles. Ο μέγιστος αριθμός αντικειμένων που δημιουργούνται εξαρτάται από το particle level που έχει ορίσει ο χρήστης στις ρυθμίσεις του παιχνιδιού.

Για την παραγωγή του τελικού οπτικού αποτελέσματος μεγάλο ρόλο έπαιξε το BlendState.Additive σαν παράμετρος στην μέθοδο spritebatch.begin(). Με την βοήθεια αυτού, τα pixels των particles παίρνουν σαν τιμή το χρωματικό άθροισμα που προκύπτει όταν ένα particle sprite βρίσκετε πάνω από ένα άλλο. Σε αντίθετη περίπτωση τα χρώματα των εξωτερικών sprites απλά θα κάλυπταν αυτά που βρίσκονται από κάτω.

Νερό και Κύματα

Για την απεικόνιση των κυμάτων πρώτα πρέπει να δημιουργήσουμε την επιφάνεια σαν μια σειρά από «ελατήρια» όπως στην εικόνα



Για την κίνηση και την συμπεριφορά αυτών των ελατηρίων εφαρμόστηκε ο νόμος του Hooke. Για να πάρουμε την κίνηση που πρέπει να αποδοθεί σε αυτά χρησιμοποίησα τον δεύτερο κανόνα κίνησης του Newton. Τέλος με αριθμητική ολοκλήρωση και τον μέθοδο Euler ήταν δυνατή η εξομοίωση κύματος. Έχοντας την επιθυμητή επιφάνεια δεν έμενε παρά η συμπλήρωση του υπόλοιπου κομματιού της οθόνης με τις κατάλληλες αποχρώσεις ώστε να δίνει την αίσθηση του νερού. Η δημιουργία και ο χειρισμός των σταγόνων γίνεται από ένα particle engine ειδικά φτιαγμένο για αυτόν τον σκοπό.

Αποθήκευση Δεδομένων και Ασύγχρονες Μέθοδοι

Το παιχνίδι έχει την δυνατότητα να αποθηκεύει στον δίσκο τα ανώτερα σκορ που έχουν επιτευχθεί και τις ρυθμίσεις που έχει επιλέξει ο χρήστης. Η αποθήκευση αυτή γίνεται με συναρτήσεις της C# φτιαγμένες για την δημιουργία και την επεξεργασία αρχείων, σε μορφή XML. Η διαδικασία της εγγραφής στον δίσκο δεν αποτελεί όμως κομμάτι της λογικής του παιχνιδιού και έτσι δεν πρέπει να παρεμβάλλεται και να καθυστερεί την εκτέλεση του. Την λύση για αυτό το πρόβλημα δίνουν οι ασύγχρονες μέθοδοι.

Από την εκκίνηση της εκτέλεσης μιας απλής μεθόδου, όλες οι εντολές της εκτελούνται στο ίδιο νήμα που έγινε η κλήση της. Αυτό έχει σαν αποτέλεσμα το νήμα να είναι καταλυμένο μέχρι το τέλος της εκτέλεσης της μεθόδου, κάτι που αποτελεί πρόβλημα για διαδικασίες που μπορεί να καθυστερήσουν. Για αυτόν τον σκοπό η επεξεργασία των αρχείων για την αποθήκευση και ανάκτηση πληροφοριών γίνετε με την κλήση ασύγχρονων μεθόδων, που δεν διακόπτουν την εξέλιξη του προγράμματος.

```
public void InitiateLoad(int mode)
        {
            currentmode = mode;
            if (mode == 1)
                currentfilename = filename;
            else if (mode == 2)
                currentfilename = filename2;
            if (!Guide.IsVisible)
            {
                if (!savingORloading)
                {
                    device = null;
                    savingORloading = true;
                    state = 2;
                    StorageDevice.BeginShowSelector(PlayerIndex.One,
this.LoadFromDevice, null);
                    loadpending = false;
                }
                else
                    loadpending = true;
            }
        }

        void LoadFromDevice(IAsyncResult result)
        {
            device = StorageDevice.EndShowSelector(result);
```

```
            IAsyncResult r = device.BeginOpenContainer(containerName, null,
null);
            result.AsyncWaitHandle.WaitOne();
            StorageContainer container = device.EndOpenContainer(r);
            result.AsyncWaitHandle.Close();
            if (container.FileExists(currentfilename))
            {
                Stream stream = container.OpenFile(currentfilename,
FileMode.Open);
                XmlSerializer serializer = new
XmlSerializer(typeof(SaveGamestr));
                try
                {
                    SaveData = (SaveGamestr)serializer.Deserialize(stream);
                    Console.WriteLine("no error");
                    stream.Close();
                    container.Dispose();
                    Array.Sort(SaveData.score);
                    Array.Reverse(SaveData.score);
                }
                catch
                {
                    Console.WriteLine("file was corrupted and its deleted");
                    stream.Close();
                    container.Dispose();
                    container.DeleteFile(currentfilename);
                }
            }
            savingORloading = false;
            state = 0;
                                        }
```

Ο παραπάνω κώδικας είναι ένα κομμάτι της διαδικασίας για την ανάκτηση των σκορ από το αρχείο. Κατά την εκκίνηση της εκτέλεσης αυτής της συνάρτησης το πρόγραμμα θα προχωρήσει κανονικά στην επόμενη εντολή, πριν δηλαδή την λήξη της, με αποτέλεσμα να μην διακοπή η συνέχεια του παιχνιδιού.

Κώδικας HLSL και Επεξεργασία της Εικόνας

Ο κώδικας που ακολουθεί είναι γραμμένος σε HLSL και αποτελεί το περιεχόμενο του αρχείου fx που είναι υπεύθυνο για την παραγωγή του εφέ της μαύρης τρύπας. Κατά την κλήση του ο κώδικας παίρνει σαν παραμέτρους το μέγεθος του παραθύρου, το κέντρο της μαύρης τρύπας (δηλαδή το επίκεντρο της επεξεργασίας της εικόνας) και την απόσταση της από την κάμερα. Έπειτα ένας pixel shader έκδοσης 4,0 επεξεργάζεται τα pixels του παραθύρου και επιστρέφει το τελικό χρώμα του κάθε pixel.

```
sampler ScreenS : register(s0);
Texture2D  myTex2D;


float2 screenSize;
float2 centerCoord;
float  distanceFromCam;

float4 BlackHolePS(float4 poz : SV_POSITION, float4 color1 : COLOR0, float2
texCoord : TEXCOORD0, float2 coords : TEXCOORD0) : SV_TARGET0
{
	centerCoord.x = centerCoord.x / screenSize.x;
	centerCoord.y = centerCoord.y / screenSize.y;
	float aspect = screenSize.x / screenSize.y;
	float4 color;

	float2 balanced = texCoord - centerCoord;
		balanced.y /= aspect;
	float2 normalized = normalize(balanced);
		float distance = length(balanced);

	float2 pos = texCoord.xy;
		pos.x = pos.x * screenSize.x;
	pos.y = pos.y * screenSize.y;
	float scaled = distance * distanceFromCam;
	float strength = 1 / (scaled * scaled);
	float3 rayDirection = float3(0, 0, 1);
		float3 surfaceNormal = normalize(float3(normalized, 1.0 /
strength));
		float3 newBeam = refract(rayDirection, surfaceNormal, 1.2);
		float2 newPos = pos + float2(newBeam.x, newBeam.y) * 300;
		color = tex2D(ScreenS, (newPos) / screenSize);
	color *= length(newBeam);
	color.a = 1.0f;

	return color;
}

technique
{
    pass P0
    {
		PixelShader = compile ps_4_0_level_9_3 BlackHolePS();
    }
                                        }
```

Στην εφαρμογή γίνεται επίσης και χρήση Bloom Effect, μιας πολύ διαδεδομένης τεχνικής με στόχο την καλύτερη και ρεαλιστικότερη απεικόνιση. Για την εφαρμογή του Bloom Effect υπάρχουν τρεις pixel shaders με αισθητά μεγαλύτερο όγκο κώδικα και πολυπλοκότητας. Τα αποτελέσματα της χρήσης του Bloom Effect φαίνονται στην παρακάτω εικόνα.

Για την επεξεργασία του τελικού αποτελέσματος είναι προτιμότερη η χρήση custom Render Target.Ο Render Target είναι ο καμβάς που έχει σαν στόχο το πρόγραμμα για την εκτύπωση της εικόνας. Ο βασικός Render Target δεν είναι άλλος από τον buffer της κάρτας γραφικών, όπου από εκεί στέλνεται η εικόνα στην οθόνη. Μπορεί όμως να δημιουργηθεί εικονικός Render Target με την μορφή ενός Texture. Έτσι μπορούμε να εκτυπώσουμε μια πρώτη φάση της εικόνας μας πάνω σε αυτό το texture, να το επεξεργαστούμε και μετά να παραδώσουμε το τελικό αποτέλεσμα στον buffer. Αυτό είναι ιδιαίτερα χρήσιμο όταν σκοπεύουμε να εφαρμόσουμε οπτικές τεχνικές όπως shader effects πάνω σε πολλά αντικείμενα. Σε αυτήν την περίπτωση εκτυπώνουμε την εικόνα πριν την επεξεργασία σε ένα texture αντί για την οθόνη, επεξεργαζόμαστε την εικόνα (δηλαδή ένα αντικείμενο) και στο τέλος δίνουμε το περιεχόμενο αυτού του texture στον buffer της κάρτας γραφικών και αυτή με την σειρά της στην οθόνη.

Installer

Μαζί με την κατασκευή του προγράμματος αυτού θεώρησα αναγκαίο να δημιουργήσω και ένα σύστημα συμπίεσης της εφαρμογής για ευκολότερο διαμοιρασμό και μεταφορά του. Το συνολικό μέγεθος που καταλαμβάνει στον δίσκο του Η/Υ μετά την εγκατάσταση του είναι περίπου 500MB ενώ το αρχείο που απαιτείται για την εγκατάσταση είναι περίπου 50MB.

Για την δημιουργία αυτού του αρχείου έκανα χρήση της εφαρμογής Inno Setup. Το Inno Setup είναι ένα πρόγραμμα που αναλαμβάνει την

δημιουργία εκτελέσιμου αρχείου εγκατάστασης με πλούσιο περιεχόμενο και δυνατότητες. Αυτό όμως δεν ήταν αρκετό. Το Content Pipeline του XNA φροντίζει τα αρχεία που δημιουργεί και διαχειρίζεται να μην είναι συμπιεσμένα, με αποτέλεσμα να καταλαμβάνουν πολύ χώρο. Για παράδειγμα το The Beautiful Blue Danube Waltz που ακούγεται κατά την διάρκεια του παιχνιδιού έχει σαν μέγεθος 6,3 MB σε μορφή mp3, ενώ το αρχείο που δημιουργεί και διατηρεί το XNA έχει μέγεθος 70,3MB σε μορφή xnb (που είναι το ίδιο μέγεθος με την έκδοση του σε μορφή wav).

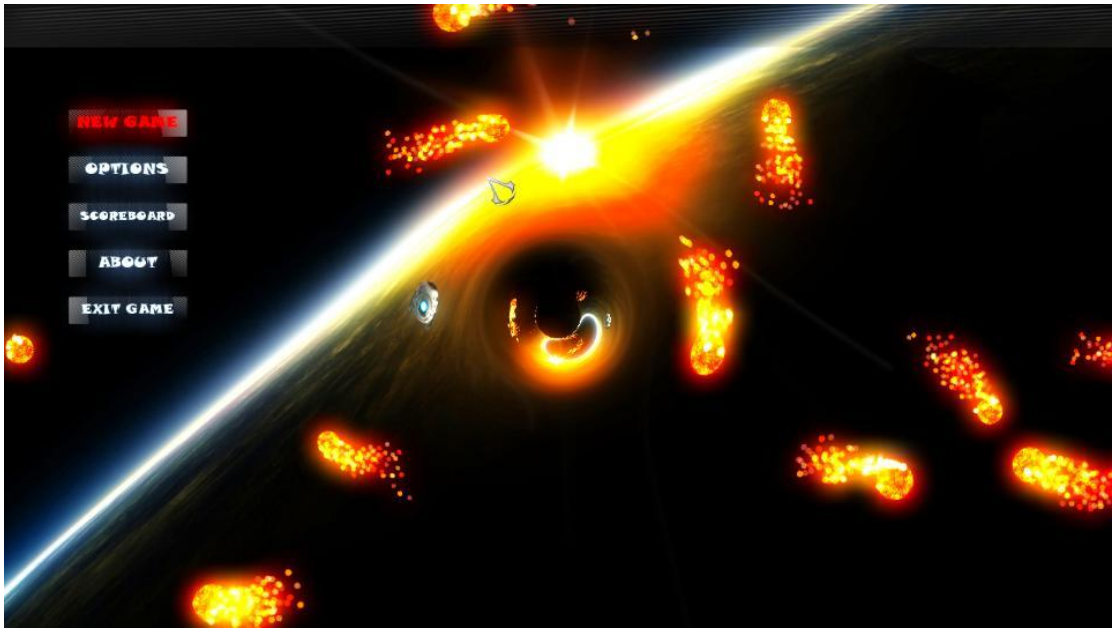
Για να λύσω αυτό το πρόβλημα κατασκεύασα μια νέα εφαρμογή σε μορφή κονσόλας. Σε αυτήν την εφαρμογή κατάφερα να απομονώσω των κώδικα εκτέλεσης του Content Pipeline χρησιμοποιώντας τον για την μετατροπή σε xnb αρχείων των contents του παιχνιδιού. Έτσι στην δημιουργία του αρχείου εγκατάστασης του παιχνιδιού υπάρχουν οι συμπιεσμένες μορφές των αρχείων, όπως mp3 και jpg όπου κατά την διάρκεια της εγκατάστασης μετατρέπονται μέσο του κώδικα της κονσόλας σε μορφή xnb και τοποθετούνται εκεί που πρέπει ώστε να τα χρησιμοποιήσει το παιχνίδι.

Screenshots με Σενάρια Χρήσης

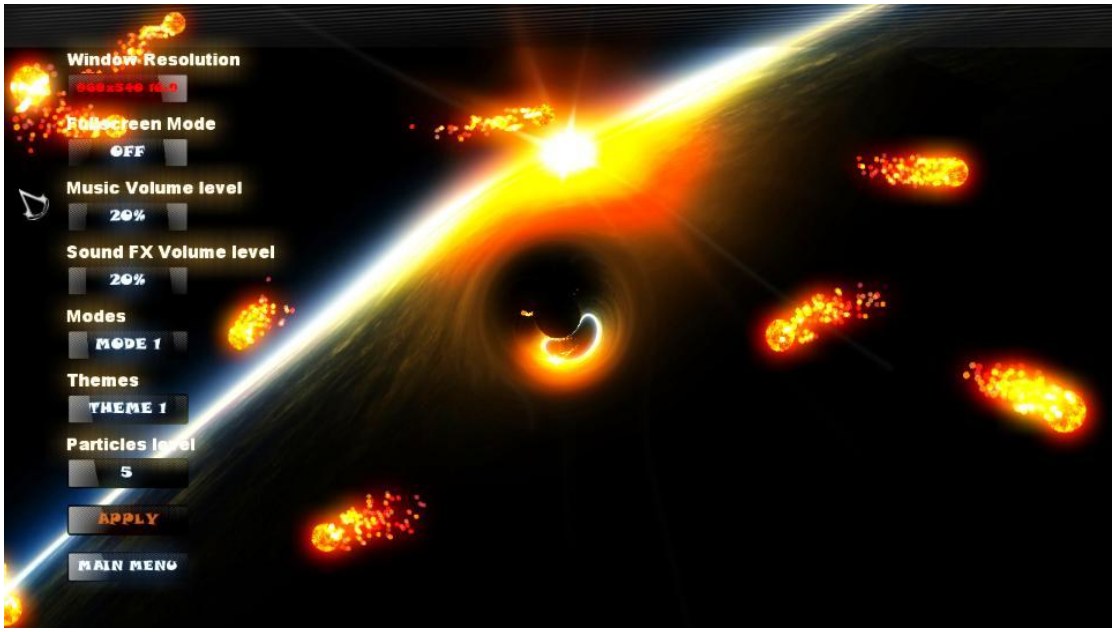
1. Το αρχικό μενού του παιχνιδιού ενώ στο background φαίνεται το mode 1.
2. Η οθόνη των ρυθμίσεων του παιχνιδιού.
3. Τα ανώτερα σκορ του mode 1.
4. Οι οδηγίες για το πώς παίζετε το mode 1.
5. Το περιεχόμενο του mode 1. Μπορούμε να δούμε ότι η μαύρη τρύπα έχει τον αριθμό 4 και στο πάνω μέρος της οθόνης φαίνεται το συνολικό σκορ των 44 πόντων.
6. Game Over. Ο χρήστης έχασε παίζοντας το mode 1 με σκορ 61
7. Εικόνα του αρχικού μενού με το επιλεγμένο το mode 2 αυτή τη φορά.
8. Η οθόνη των μεγαλύτερων σκορ του mode 2.
9. Η οθόνη με τις οδηγίες για το πώς παίζετε το mode 2.

10. Εισαγωγή στο παιχνίδι με έναν παίκτη.
11. Εισαγωγή στο παιχνίδι με δύο παίκτες.
12. Το περιεχόμενο του mode 2. Εδώ ο βλέπουμε μια τορπίλη έτυμη να συγκρουστεί με μία νάρκη, δύο βέλη που υποδηλώνουν τον ερχομό δύο νέων ναρκών και την πορεία τους, νάρκες σε διαδικασία αυτοκαταστροφής, ένα powerup που προσθέτει τορπίλες στο υποβρύχιο, την μπάρα ζωής του υποβρυχίου και κάτω αριστερά τα χαρακτηριστικά του.
13. Παρόμοια κατάσταση με την εικόνα 12 μόνο που τώρα το παιχνίδι βρίσκεται σε παύση από τον χρήστη (κατά την διάρκεια της οποίας αντιστρέφονται τα χρώματα των αντικειμένων). Όπως βλέπουμε κατά την διάρκεια της παύσης ο χρήστης έχει πρόσβαση στο μενού του παιχνιδιού.
14. Ένα καρέ του animation της έκρηξης τριών ναρκών.
15. Ένα καρέ με το υποβρύχιο να έχει δεχθεί μεγάλη ζημιά
16. Εδώ φαίνεται η ειδική τορπίλη κατά την εκτόξευση της όπως στρέφετε με σκοπό να συγκρουστεί με την νάρκη που θα προσφέρει περισσότερους πόντους στον παίκτη.
17. Η οθόνη επιλογής παικτών με την χρήση joypad.
18. Η εκκίνηση του mode 2 με δύο παίκτες.
19. Ένα ακόμα καρέ του mode 2 με δύο παίκτες κατά την διάρκεια του παιχνιδιού.

Εικόνα 1

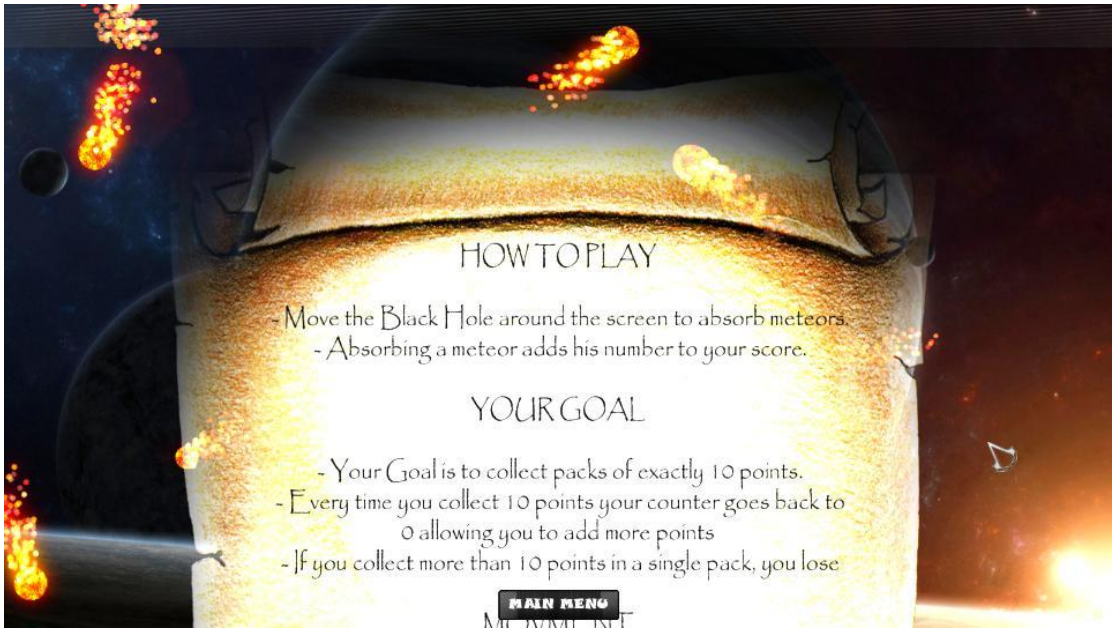


Εικόνα 2



Εικόνα 3

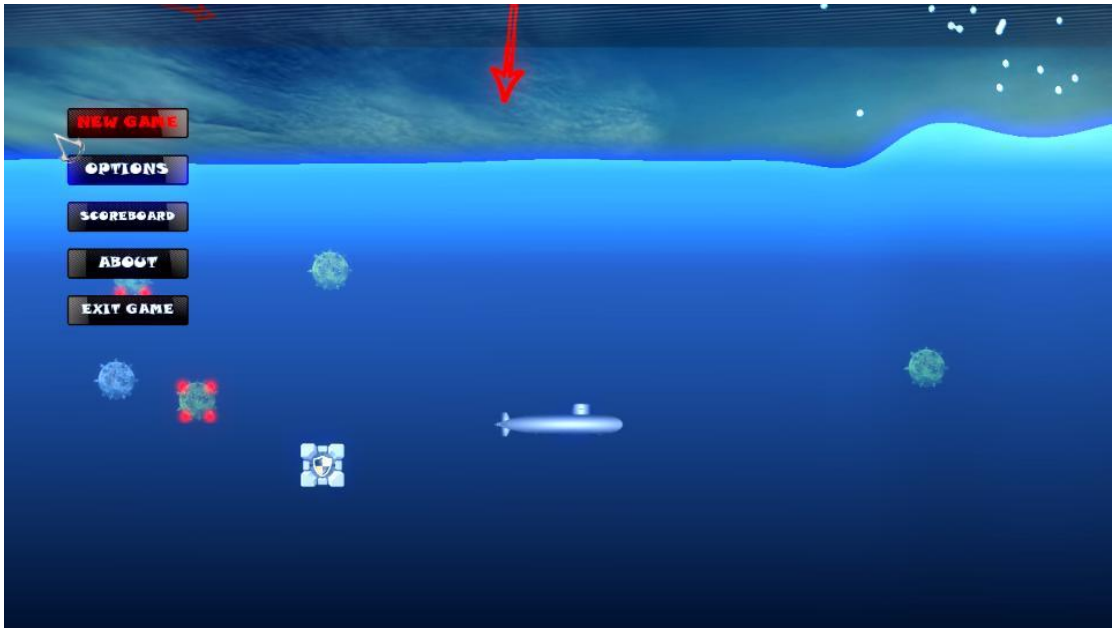
Εικόνα 4



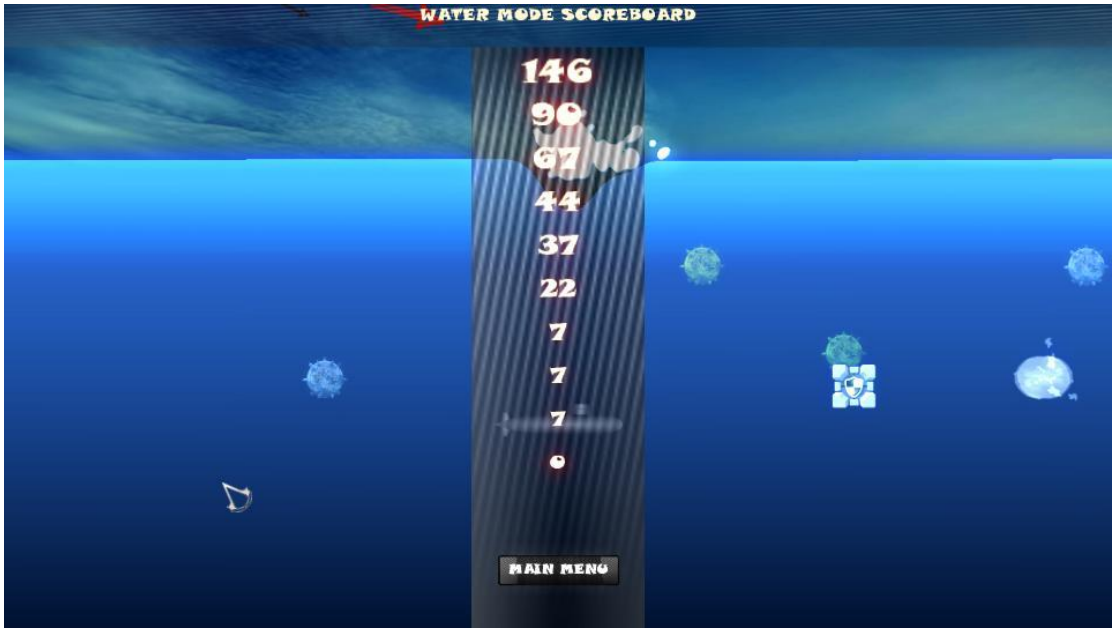
Εικόνα 5

Εικόνα 6

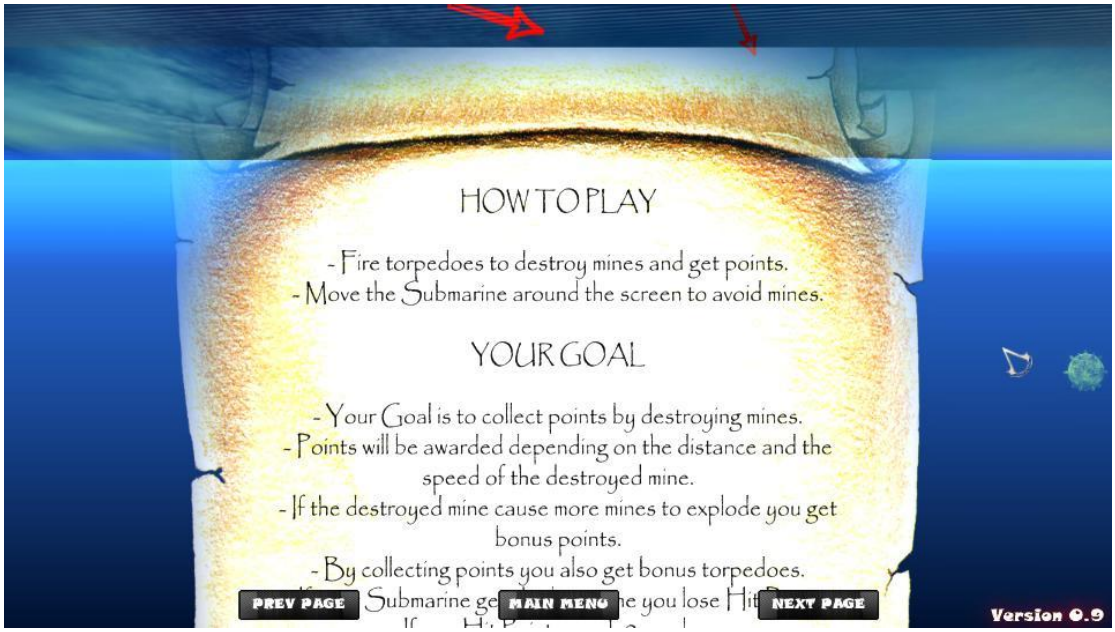
Εικόνα 7



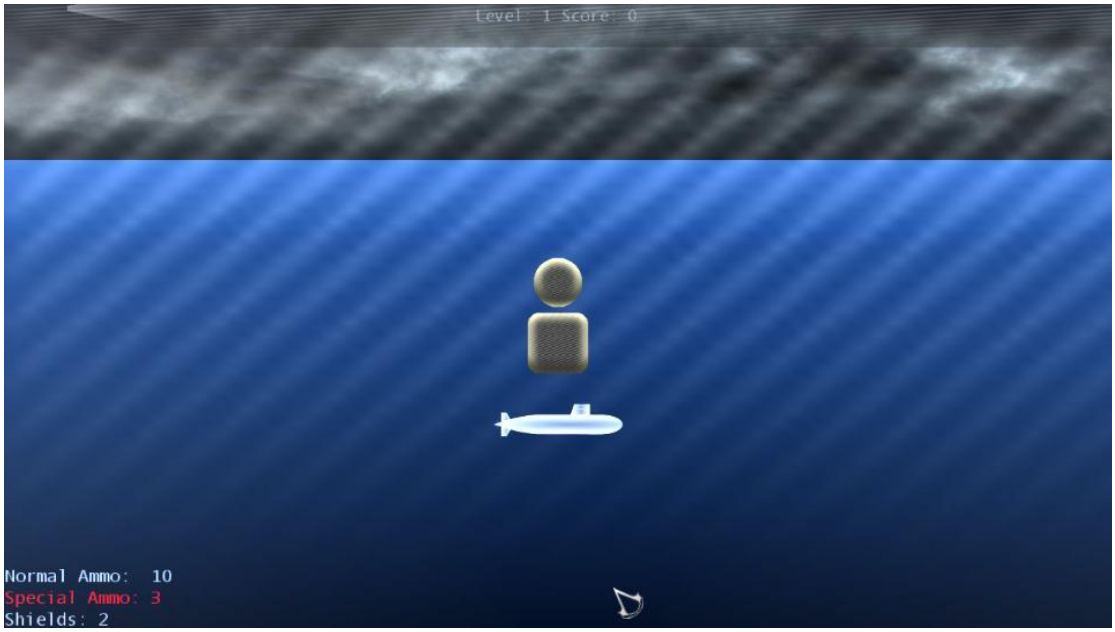
Εικόνα 8



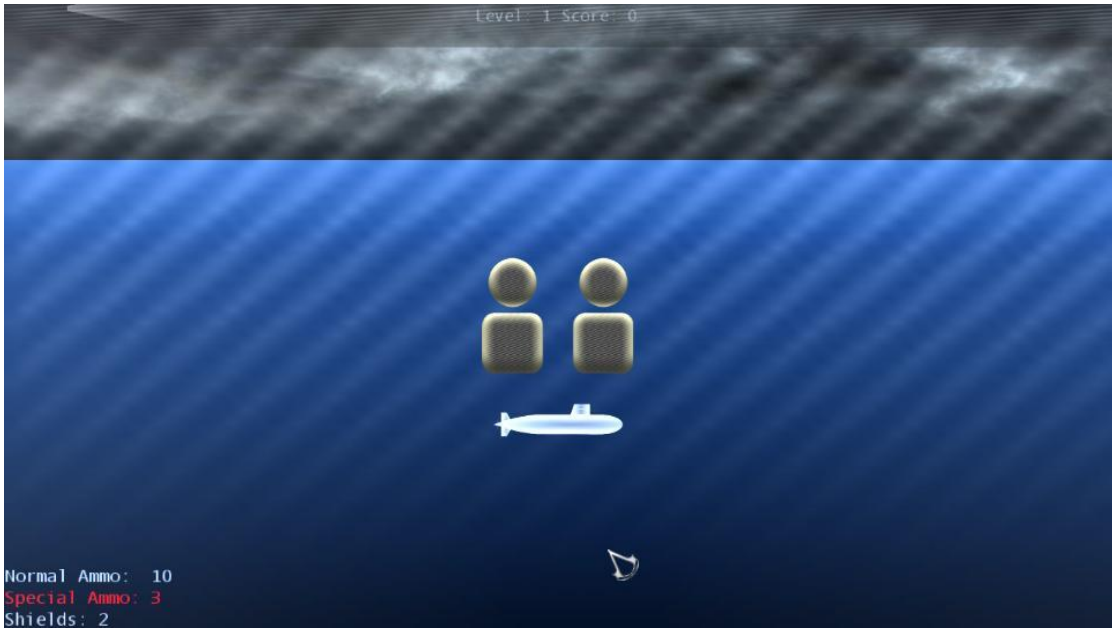
Εικόνα 9



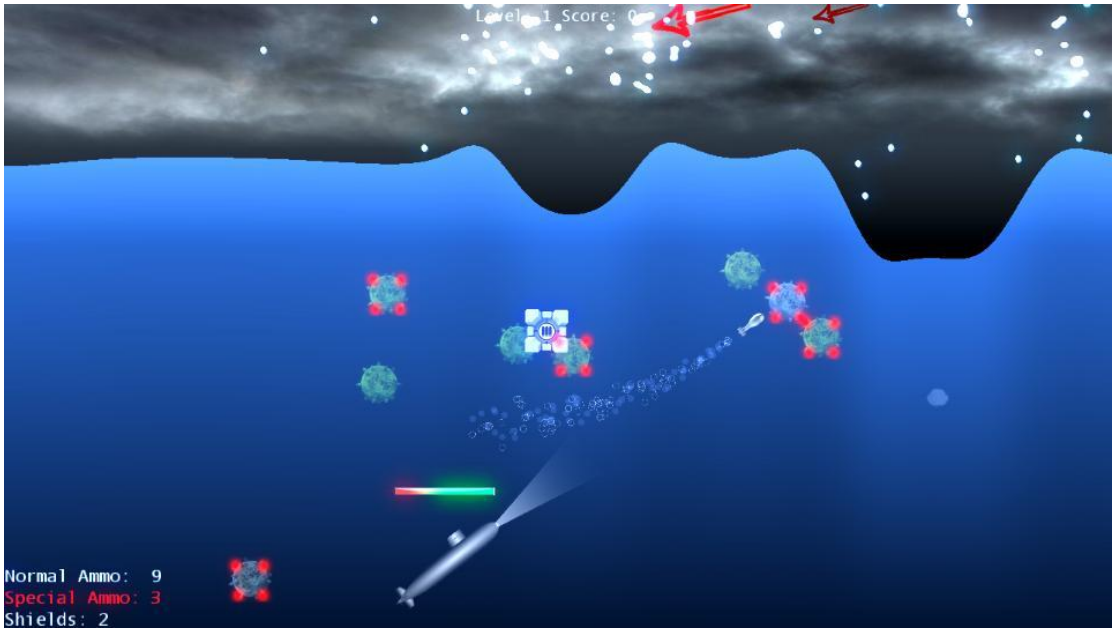
Εικόνα 10



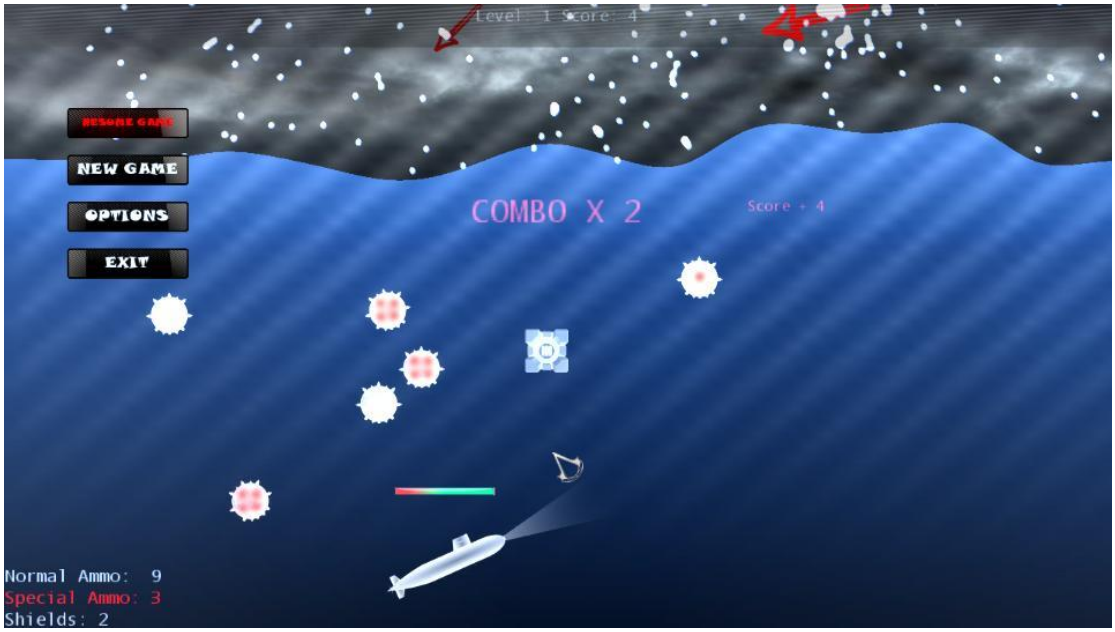
Εικόνα 11



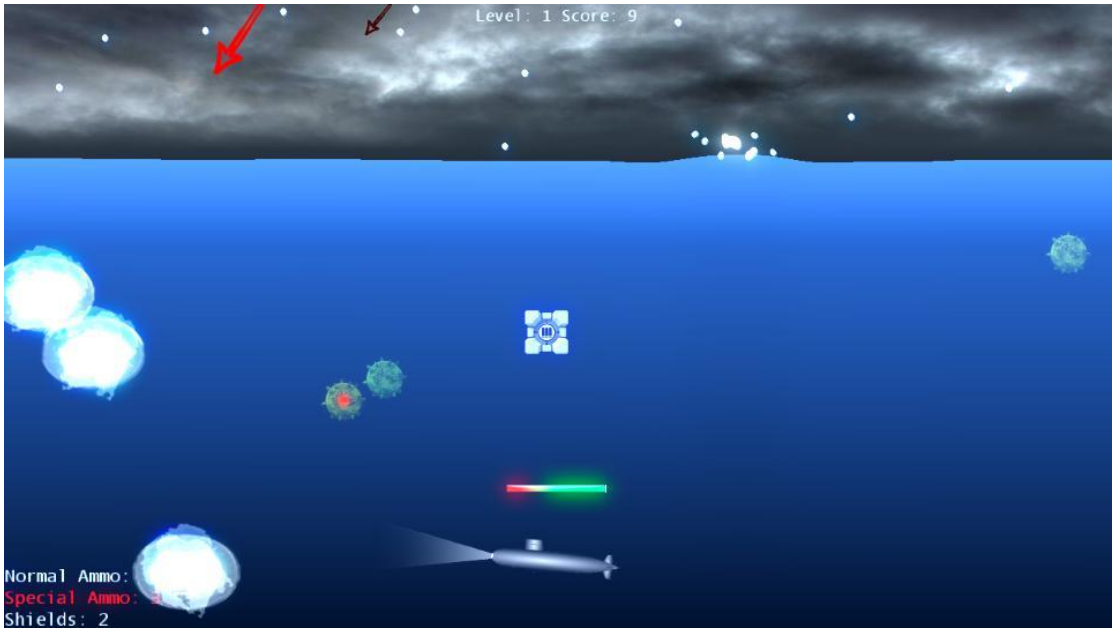
Εικόνα 12



Εικόνα 13



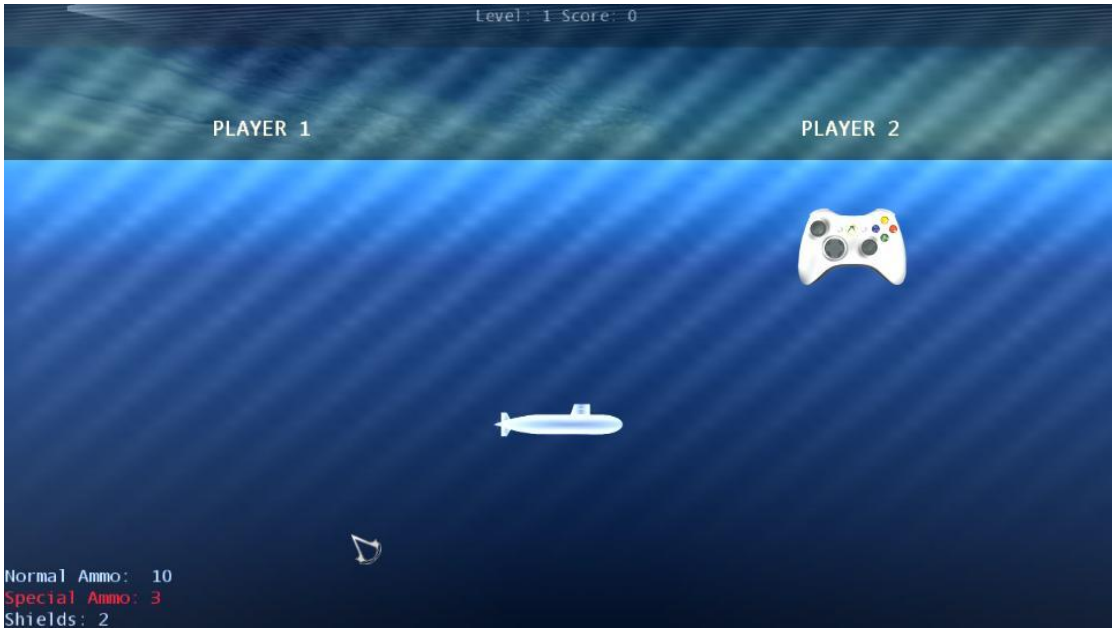
Εικόνα 14



Εικόνα 15

Εικόνα 16

Εικόνα 17



Εικόνα 18

Εικόνα 19


Player 1 score: 0
Level: 1
Player 2 score: 0
Normal Ammo: 10
Special Ammo: 3
Shields: 2
Normal Ammo: 10
Special Ammo: 3
Shields: 2