ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΣΧΕΔΙΑΣΗ & ΥΛΟΠΟΙΗΣΗ ΠΛΑΦΟΡΜΑΣ ΑΝΑΠΤΥΞΗΣ VIDEO GAME

-

DESIGN & IMPEMENTATION OF A GAME ENGINE

ΠΡΟΠΤΥΧΙΑΚΟΣ ΦΟΙΤΗΤΗΣ
ΑΝΑΣΤΑΣΟΠΟΥΛΟΣ ΒΑΣΙΛΕΙΟΣ - 727

ΕΠΙΒΛΕΠΟΥΣΑ ΚΑΘΗΓΗΤΡΙΑ
ΤΣΟΜΠΑΝΟΠΟΥΛΟΥ ΠΑΝΑΓΙΩΤΑ

ΠΕΡΙΕΧΟΜΕΝΑ

1. Εισαγωγή στα video games

1.1 Εισαγωγή

Τα τελευταία χρόνια, τα βιντεοπαιχνίδια έχουν εισέλθει για τα καλά στη ζωή μας, προσφέροντας δυνατές συγκινήσεις και ενθουσιασμό σε παιδιά και εφήβους. Πολλές φορές βέβαια τα "παιδιά" αυτά τυγχάνει να είναι και μεγαλύτερης ηλικίας. Τα video games είναι μια σχετικά νέα καινοτομία. Έχουν υπάρξει σε διάφορες μορφές από την έναρξη των ηλεκτρονικών υπολογιστών και με πολλούς τρόπους ήταν απαραίτητα στη διαδρομή που ακολούθησαν οι υπολογιστές. Συχνά ενοχοποιούνται για τη βία τους ή τη δύναμη του εθισμού τους, ωστόσο τα ηλεκτρονικά παιχνίδια μπορούν επίσης, σύμφωνα με ερευνητές και τους εμπνευστές των βιντεοπαιχνιδιών, να έχουν ευεργετικές συνέπειες στην υγεία, στην εκπαίδευση και σε άλλα κοινωνικά προβλήματα.

1.2 Ορισμός του video game

Τόσο στην βιβλιογραφία όσο και στο διαδίκτυο υπάρχουν πολλοί ορισμοί για τα video games. Ο πιο δόκιμος κατά την γνώμη μου είναι ο παρακάτω:

«Ο όρος video game αναφέρεται σε κάθε είδους ηλεκτρονικό παιχνίδι το οποίο περιλαμβάνει την ανθρώπινη αλληλεπίδραση με μια διεπαφή χρήστη (user interface), για την παραγωγή οπτικής ανάδρασης σε μια συσκευή απεικόνισης.»

1.3 Θεωρία του διασκεδαστικού

Τα παιχνίδια υπολογιστών μπορούν να ταξινομηθούν σε διάφορες κατηγορίες, όπως περιπέτειας, δράσης, τρόμου κτλ, όμως όλα κατά βάση έχουν ένα κοινό στόχο, να διασκεδάσουν τον χρήστη. Τι είναι όμως αυτό που κάνει μία δραστηριότητα διασκεδαστική;

Οι άνθρωποι από την φύση μας, είμαστε καταπληκτικές μηχανές εύρεσης προτύπων (patterns). Καθημερινά ψάχνουμε να βρούμε συνέχεια τέτοια πρότυπα είτε σε ανθρώπινες συμπεριφορές, είτε σε φυσικά φαινόμενα ή γενικότερα σε οτιδήποτε υπάρχει γύρω μας, ακόμα και αν αυτά τα πρότυπα δεν υπάρχουν. Όταν τα πράγματα και οι καταστάσεις γίνονται πολύ περίπλοκες και δεν μπορούμε να βρούμε ή να φτιάξουμε τέτοια πρότυπα εκνευριζόμαστε και τα παρατάμε. Όταν τα εντοπίσουμε

μας αρέσει να τα αποκρυπτογραφούμε και να τα βλέπουμε να συμβαίνουν ξανά και ξανά.

Το να εκπαιδεύουμε το μυαλό μας είναι κάτι διασκεδαστικό. Τα παιχνίδια μας προσφέρουν αυτή την δυνατότητα μέσα από την επίλυση γρίφων, κάτι το οποίο απαιτεί σκέψη και εκπαίδευση στο να αναλύουμε πρότυπα. Παίζουμε ένα παιχνίδι μέχρι να το κατανοήσουμε σε μεγάλο βαθμό και να γίνουμε πολύ καλοί σε αυτό. Κατανοώντας τα πρότυπα και δημιουργώντας έναν ιδανικό τρόπο παιξίματος, κάνουμε το παιχνίδι όσο το δυνατόν πιο προβλέψιμο κάτι το οποίο τελικά μας κάνει να το βαρεθούμε και να μην το βρίσκουμε πλέον διασκεδαστικό. Στον πραγματικό κόσμο αυτό το ονομάζουμε "ασφάλεια", "σταθερή δουλειά" ή "ρουτίνα", γιατί ο εγκέφαλος, άρα και κατ επέκταση ο άνθρωπος, πάντα θα προσπαθεί να βελτιστοποιεί, να απλοποιεί και να προγραμματίζει τις ενέργειες του προκειμένου να καλύψει τους στόχους του και τις ανάγκες του όσο το δυνατόν σε μεγαλύτερο βαθμό.

Το σκάκι για παράδειγμα είναι ένα παιχνίδι που υπάρχει πάνω από 1500 χρόνια και παρόλα αυτά συναρπάζει μικρούς και μεγάλους μέχρι σήμερα λόγω της πολυπλοκότητας και του πλήθους των γρίφων που μπορεί να προσφέρει. Με άλλα λόγια στο σκάκι μπορούμε να βρούμε κάποια πρότυπα ως προς τις κινήσεις που πρέπει να κάνουμε σε ορισμένες περιπτώσεις, αλλά είναι αδύνατον να βρούμε ένα ιδανικό τρόπο παιξίματος για όλες τις πιθανές εκβάσεις του παιχνιδιού. Γι αυτό το λόγο είναι ένα παιχνίδι το οποίο ή θα λατρέψει κάποιος όταν το κατανοήσει ή θα του φανεί ανούσιο και βαρετό, επειδή δεν μπόρεσε ποτέ να καταλάβει την λογική του παιχνιδιού. Αν μπορούσαμε να δώσουμε έναν ορισμό στο τι είναι διασκεδαστικό αυτός θα ήταν:

«Διασκεδαστική είναι η διαδικασία ανακάλυψης περιοχών σε έναν πιθανοτικό χώρο.»

Ο στόχος λοιπόν των σχεδιαστών παιχνιδιών είναι να αυξήσουν τον πιθανοτικό χώρο εξερεύνησης ούτος ώστε να κρατήσουν το ενδιαφέρον του χρήστη αμείωτο. Γίνονται λοιπόν πολλές προσπάθειες είτε με αναδυόμενο gameplay, είτε με μη γραμμική εξέλιξη της ιστορίας του παιχνιδιού προκειμένου να γρίφοι που εμφανίζονται να είναι κάθε φορά διαφορετικοί. Προσπάθειες για παιχνίδια στα οποία οι γρίφοι θα επιδέχονται παραπάνω από μία λύση, γρίφοι οι οποίοι προσφέρονται προς ερμηνεία. Τότε λέμε ότι ένα παιχνίδι είναι τέχνη, όταν δηλαδή αυτό γίνεται αντικείμενο προς ερμηνεία.

2. Σχεδίαση ενός Video Game

1.1 Εισαγωγή

Ένα video game είναι κάτι περισσότερο από τον συνδυασμό των επιμέρους κομματιών του. Είναι μια συνισταμένη ενεργειών και δράσεων, που όταν το παιχνίδι ολοκληρωθεί το κάνει κάτι μοναδικό. Η δημιουργία αυτής της συνέργειας απαιτεί τεχνογνωσία, καθώς και μία αίσθηση σχεδιασμού και της τέχνης. Ο σχεδιασμός ενός παιχνιδιού ορίζεται ως η διαδικασία της δημιουργίας της δομής και της φύσης του παιχνιδιού καθώς και ο τρόπος με τον οποίο λειτουργεί. Αυτό περιλαμβάνει τους κανόνες και τους αλγόριθμους που αποτελούν το θεμέλιο του σχεδιασμού παιχνιδιών, αλλά και πιο απόκρυφα χαρακτηριστικά, όπως η διεπαφή που περιγράφει το πώς ο παίκτης του παιχνιδιού επικοινωνεί με το παιχνίδι αλλά και το πώς το παιχνίδι επικοινωνεί με το παίκτη.

Χρησιμοποιώντας το προηγούμενο παράδειγμα, στο σκάκι οι τρόποι με τους οποίους τα κομμάτια κινούνται, η εναλλασσόμενη σειρά που παίζουν οι δύο αντίπαλοι, οι κανόνες σχετικά με την κίνηση και την αιχμαλώτιση των πιονιών, η δυνατότητα δημιουργίας νέας βασίλισσας από απλά πιόνια και η λήξη του παιχνιδιού (checkmate), είναι όλα μέρος του σχεδιασμού. Επιπλέον, η 8x8 διάταξη της σκακιέρας είναι και αυτή μέρος του σχεδιασμού, καθώς και το σχήμα των πιονιών και η ονομασία τους. Τα στοιχεία αυτά αποτελούν αναπόσπαστο κομμάτι του παιχνιδιού καθώς χωρίς αυτά στην ουσία δεν υπάρχει παιχνίδι.

Σε πρώτη φάση, ο σχεδιασμός ενός παιχνιδιού είναι μια πάρα πολύ σύνθετη δραστηριότητα για να μπορέσει να αναχθεί σε μια μοντελοποιημένη διαδικασία. Επιπλέον, η προσωπικότητα του σχεδιαστή του παιχνιδιού, επηρεάζει εκ των πραγμάτων τις μεθόδους που αυτός χρησιμοποιεί. Έτσι η διαδικασία του σχεδιασμού μπορεί να ακολουθήσει κάποιες γενικές κατευθυντήριες γραμμές, αλλά σε καμία περίπτωση δεν μπορεί να αποτελέσει μία αυτοματοποιημένη και τυπική φόρμουλα.

2.2 Επιλογή ιδέας και στόχων

Πριν ξεκινήσουμε την διαδικασία δημιουργίας ενός παιχνιδιού, θα πρέπει πρώτα να βρούμε πρώτα μια ιδέα, μια ιστορία, εμπνευσμένη για κάτι πρωτότυπο κάτι μοναδικό. Η ιδέα του παιχνιδιού είναι μια συγκεκριμένη συλλογή κανόνων και γεγονότων μέσω των οποίων θα ορισθεί ο πρωταρχικός στόχος.

Ένα παιχνίδι πρέπει να έχει ένα σαφώς καθορισμένο στόχο. Αυτός ο στόχος πρέπει να ορίζεται και να περιγράφεται σε σχέση με την επίδραση που θα έχει στον παίκτη. Ο πρωταρχικοί στόχοι των παιχνιδιών ποικίλουν και μπορεί να είναι η μέγιστη συλλογή πόντων (high score) ή η εξερεύνηση ενός τόπου χωρίς να ξεμείνει ο παίχτης από ζωές. Επίσης δεν είναι απαραίτητο να υπάρχει ένας και μοναδικός στόχος στο παιχνίδι, μπορεί ο σχεδιαστής να θέσει έναν συνδυασμό από στόχους. Πολλοί σχεδιαστές παιχνιδιών μπορεί να ακολουθήσουν και την αντίστροφη διαδικασία σχεδίασης. Να ξεκινήσουν δηλαδή με την επιλογή των στόχων και το θέμα του παιχνιδιού να εξαρτάται από αυτούς.

Ένας από τους καλύτερους τρόπους για να ξεκινήσουμε την δημιουργία ενός παιχνιδιού είναι να φτιάξουμε ένα storyboard, να δημιουργήσουμε δηλαδή μια σειρά από σχέδια και εικόνες που θα δείχνουν τα επίπεδα του παιχνιδιού και τις διαφορετικές σκηνές και τις αποστολές που καλείται ο παίκτης να φέρει εις πέρας. Κάθε storyboard κάλο είναι να περιλαμβάνει μια παράγραφο ή μερικά αποσπασματικά σχόλια του τι συμβαίνει σε κάθε σκηνή του παιχνιδιού έτσι ώστε αυτό να γίνεται αντιληπτό και ξεκάθαρο και στα υπόλοιπα μέλη της ομάδας.

2.3 Έρευνα και προετοιμασία

Με την κύρια ιδέα και το στόχο ξεκαθαρισμένα και τοποθετημένα καλά στο μυαλό μας, το επόμενο βήμα είναι η ανάπτυξη του θέματος. Το πρώτο βήμα περιλαμβάνει την συγκέντρωση όσο το δυνατόν περισσότερων στοιχείων σχετικά με το θέμα. Στην συνέχεια ακολουθεί η μελέτη όλων των προηγούμενων σκέψεων και ιδεών του σχεδιαστή που συνδέονται με τον στόχο ή το θέμα του παιχνιδιού. Τέλος, ο θα πρέπει να βεβαιωθούμε ότι υπάρχει άριστη κατανόηση από μέρους του, σχετικά με το περιβάλλον που το παιχνίδι θα προσπαθήσει να αναπαραστήσει. Το παιχνίδι θα πρέπει να δίνει μια αίσθηση αυθεντικότητας και να μεταδίδει την υφή του πραγματικού κόσμου, δύο συνισταμένες που μπορούν να επιτευχθούν μόνο με την καλή κατανόηση του περιβάλλοντος του παιχνιδιού.

2.4 Κύριες δομές του παιχνιδιού

Πρωταρχικός στόχος στην φάση του σχεδιασμού είναι η δημιουργία διαγράμματος των τριών αλληλοεξαρτώμενων δομών του παιχνιδιού, δηλαδή της διεπαφής ή αλλιώς I/O structure, της δομής του παιχνιδιού σαν ιδέα και κανόνες, γνωστό και ως Game structure και της δομής του παιχνιδιού σαν υπολογιστικό πρόγραμμα, Program structure. Το I/O structure είναι το σύστημα μέσω του οποίου μεταφέρονται πληροφορίες από τον παίχτη στον υπολογιστή και αντίστροφα. Game structure είναι η εσωτερική αρχιτεκτονική των αιτιωδών σχέσεων που καθορίζουν τα εμπόδια που ο παίκτης πρέπει να ξεπεράσει κατά τη διάρκεια του παιχνιδιού. Program structure, είναι η οργάνωση του προγραμματιστικού κώδικα και των αλγορίθμων εκείνων που συνθέτουν το σύνολο του προγράμματος. Και οι τρεις δομές πρέπει να δημιουργηθούν ταυτόχρονα, καθώς πρέπει να αναπτυχθούν από κοινού. Αποφάσεις που αφορούν την μία δομή πρέπει να ελέγχονται για τις επιπτώσεις που μπορεί να έχουν στις άλλες δομές. Οι αλληλοεξαρτώμενες δομές του παιχνιδιού περιγράφονται αναλυτικότερα παρακάτω.

- **I/O structure**

Αποτελείται από δεδομένα εισόδου και εξόδου. Σε αντίθεση με τις ανθρώπινες γλώσσες, η σχέση των παραπάνω δύο δεν είναι συμμετρική. Ο υπολογιστής διαθέτει δύο μέσα παραγωγής αποτελεσμάτων για τον άνθρωπο, τα γραφικά της οθόνης και τον ήχο. Τα γραφικά θα πρέπει να επικοινωνούν δυναμικά και με συναίσθημα με τον χρήστη και να είναι κατά κύριο λόγο καλαίσθητα. Ο σχεδιασμός των γραφικών θα πρέπει να είναι λειτουργικός και με νόημα έτσι ώστε να μεταφέρουν τις κρίσιμες πληροφορίες του παιχνιδιού υποστηρίζοντας ταυτόχρονα και την φαντασία του παίχτη.

Η δομή εισόδου δεδομένων βρίσκεται στο επίκεντρο ενός θεμελιώδους διλήμματος που όλοι οι σχεδιαστές παιχνιδιών πρέπει να αντιμετωπίσουν. Ένα εξαιρετικό παιχνίδι επιτρέπει στον παίκτη να αλληλεπιδρά σε μεγάλο βαθμό με τον αντίπαλο του και να επενδύσει ένα μεγάλο μέρος της προσωπικότητάς του μέσα στο παιχνίδι. Αυτό προϋποθέτει ότι το παιχνίδι προσφέρει στον παίκτη έναν μεγάλο αριθμό ουσιαστικών επιλογών μέσω των οποίων μπορεί να εκφράσει ο παίκτης την προσωπικότητα του. Ωστόσο, οι αποφάσεις πρέπει να εισαχθούν μέσα στο παιχνίδι, και ο μεγάλος αριθμός των επιλογών φαίνεται να απαιτούν μια εκτεταμένη και περίπλοκη δομή εισόδου η οποία θα μπορούσε κάλλιστα να είναι εκφοβιστική για τον παίκτη. Η πρόκληση λοιπόν, είναι ότι ένα εξαιρετικό παιχνίδι φαίνεται να απαιτεί μια ογκώδη δομή εισόδου που όμως δεν θα πρέπει να μπερδεύει ή να κουράζει τον παίχτη.

Το I/O structure είναι ίσως από η πιο σημαντική από τις τρεις δομές σε ένα

παιχνίδι υπολογιστή, καθώς αποτελεί το πρόσωπο του παιχνιδιού που βλέπει ο παίκτης. Είναι το όχημα της αλληλεπίδρασης για το παιχνίδι. Είναι επίσης η πιο δύσκολη από τις τρεις δομές στον σχεδιασμό της, καθώς απαιτεί τόσο ανθρώπινη ευαισθησία όσο και πλήρη τεχνική γνώση του υπολογιστή.

- **Game structure**

Το κεντρικό πρόβλημα στο σχεδιασμό της δομής του παιχνιδιού είναι η ένταξη της φαντασίας του στόχου και του θέματος μέσα σε ένα λειτουργικό σύστημα. Ο σχεδιαστής του παιχνιδιού πρέπει να εντοπίσει ένα βασικό στοιχείο από το περιβάλλον και να οικοδομήσει το παιχνίδι γύρω από αυτό το βασικό στοιχείο. Αυτό το βασικό στοιχείο πρέπει να είναι κεντρικής σημασίας για το θέμα, αντιπροσωπευτικό και συμβολικό των θεμάτων που εξετάζονται σε αυτό το παιχνίδι, εύχρηστο και κατανοητό. Πολλά παιχνίδια χρησιμοποιούν πολλαπλά τέτοια στοιχεία, χωρίς αυτό να είναι απαραίτητα κακό, καθώς ένα παιχνίδι μπορεί να είναι επιτυχημένο, αρκεί όλα τα κύρια στοιχεία του να δένουν αρμονικά μεταξύ τους και να μην έρχονται σε σύγκρουση. Ωστόσο, η ύπαρξη πολλών βασικών στοιχείων αυξάνουν την πιθανότητα να παραβιαστούν κάποιες από τις αρχές του παιχνιδιού, κάτι που πολλές φορές μπορεί να αποπροσανατολίσει τον σχεδιαστή δημιουργώντας ένα παιχνίδι αρκετά πολύπλοκο και χωρίς συνοχή που ίσως αργότερα να μπερδέψουν τον παίχτη.

Το κύριο πρόβλημα κατά την δημιουργία του I/O structure είναι η εξάλειψη των εμποδίων λειτουργικότητας ενώ το κύριο πρόβλημα με την δημιουργία του game structure είναι η κατανόηση των λειτουργιών που θα προσφέρονται μέσα από το παιχνίδι. Επίσης οι σχεδιαστές θα πρέπει να επικεντρωθούν στον να δώσουν αρκετό "χρώμα" και φαντασία στο παιχνίδι τους, έτσι ώστε να μεταφέρει στον χρήστη μια αυθεντική αίσθηση της πραγματικότητας. Ένα πολύ κοινό λάθος που κάνουν οι περισσότεροι σχεδιαστές στην προσπάθεια τους να κάνουν πιο ενδιαφέρον το παιχνίδι τους, είναι να το φορτώνουν με πάρα πολλά χαρακτηριστικά και λειτουργίες, δημιουργώντας έτσι ένα χαοτικό και περίπλοκο παιχνίδι το οποίο δύσκολα θα κρατήσει το κοινό στο οποίο απευθύνεται.

- **Program structure**

Το Program structure είναι το τρίτο αντικείμενο στο οποίο θα πρέπει να δοθεί προσοχή κατά την διαδικασία του σχεδιασμού. Αυτή η δομή είναι το όχημα με το οποίο μεταφράζεται το I/O structure και το game structure σε ένα πραγματικό προϊόν. Ένα από τα πιο σημαντικά στοιχεία της δομής του προγράμματος είναι το γνωστό και ως "memory mapping". Θα πρέπει δηλαδή να σχεδιασθεί το πώς θα διανεμηθεί κατάλληλα η διαθέσιμη μνήμη σε κάθε στοιχείο και λειτουργία του παιχνιδιού,

προκειμένου το πρόγραμμά μας να εκτελείται ομαλά και χωρίς να κολλάει. Εάν δεν δώσουμε μεγάλη προσοχή στη διαχείριση της μνήμης, είναι πολύ πιθανό να καταλήξουμε στο σημείο να δαπανώνται υπερβολικά μεγάλες ποσότητες μνήμης για μικρής σημασίας λειτουργίες και να μένει ελάχιστη ή καθόλου μνήμη για τις πιο σημαντικές. Τέλος, πολύ σημαντική είναι η τεκμηρίωση της ροής του προγράμματος, δηλαδή ότι όλα λειτουργούν όπως ακριβώς σχεδιάστηκαν.

2.5 Κατασκευή του Game Design Document (GDD)

Αφού τελειώσουμε με τα storyboards και με όλες τις λεπτομέρειες του παιχνιδιού, πρέπει να δημιουργήσουμε ένα έγγραφο σχεδιασμού ή αλλιώς Game Design Document (GDD). Ο στόχος του εγγράφου είναι να καταγράψει όλες τις ιδέες και τις λειτουργίες του παιχνιδιού για να είναι σαφώς ορισμένα και ξεκάθαρα από την αρχή αυτά που θέλουμε να υλοποιήσουμε, είναι κάτι που θα μπορούσαμε να το παρομοιάσουμε με σενάριο ταινίας. Η δημιουργία του εγγράφου είναι μία δικλείδα ασφαλείας και ένας οδηγός για τις επόμενες φάσεις κατασκευής του παιχνιδιού. Το αποτέλεσμα της ανάπτυξης του εγγράφου σχεδιασμού είναι ότι ο φανταστικός κόσμος γίνεται πιο έντονος και πιο ξεκάθαρος στους υλοποιητές. Η δημιουργία μίας σαφής εικόνας του κόσμου του παιχνιδιού είναι ένα από τα πιο σημαντικά ζητήματα στο σχεδιασμό του παιχνιδιού.

Εφόσον υπάρχει ένα έγγραφο σχεδιασμού, είναι πιο εφικτό να δημιουργηθεί ένα σταθερό και ώριμο παιχνίδι. Κατά την διάρκεια της κατασκευής του παιχνιδιού δεν θα πρέπει να εισαχθούν αυθόρμητα στοιχεία που είναι έξω από τον κόσμο του παιχνιδιού που περιγράφει το GDD. Όλα τα στοιχεία και τα μέρη του παιχνιδιού θα πρέπει να λειτουργούν αρμονικά μεταξύ τους, έτσι ώστε η προσοχή των παικτών να μην αποσπάται από ανακολουθίες στον κόσμο του παιχνιδιού.

Επίσης το game design document μπορεί να διαχωριστεί σε επιμέρους έγγραφα, κάτι το οποίο κρίνεται απαραίτητο εάν το παιχνίδι είναι πολύ περίπλοκο και πολύ μεγάλο. Τα επιμέρους έγγραφα στα οποία μπορεί να χωριστεί game design document είναι τα ακόλουθα:

- Conceptual design phase: Game Overview Document (GOD)
- Initial Design Phase: Initial Design Document (IDD)
- Expanded design Phase: Expanded Design Document (EDD)
- Final Design Phase: Final Design Document (FDD)

2.6 Στάδια υλοποίησης του παιχνιδιού

Αφού τελειώσουμε με το θέμα του παιχνιδιού, τις γενικότερες ιδέες που θα το απαρτίζουν και με την κατασκευή του game design document μπορούμε να προχωρήσουμε στο επόμενη φάση ανάπτυξης του παιχνιδιού, το στάδιο της υλοποίησης, δηλαδή το κομμάτι στο οποίο η ιδέα μας θα αρχίσει να παίρνει σάρκα και οστά.

- **Στάδιο αξιολόγησης του σχεδιασμού (Evaluation of Design Phase)**

Μετά την ολοκλήρωση όλων των προηγούμενων φάσεων που περιγράφουμε παραπάνω, ακολουθεί η αξιολόγηση του συνολικού σχεδιασμού για των εντοπισμό των πιο κοινών σχεδιαστικών ελαττωμάτων που μαστίζουν τα παιχνίδια. Το πρώτο και πιο σημαντικό ερώτημα είναι αν αυτό το σχέδιο ικανοποιεί τους στόχους του σχεδιασμού. Εάν είμαστε ικανοποιημένοι και ο σχεδιασμός έχει περάσει αυτό το κρίσιμο τεστ, μπορούμε να προχωρήσουμε στην επόμενη φάση υλοποίησης. Η αξιολόγηση του σχεδιασμού είναι πολύ σημαντική, γιατί το επόμενο βήμα είναι ο προγραμματισμός του παιχνιδιού. Έτσι, πριν από τη δέσμευση για τον προγραμματισμό, θα πρέπει πρώτα να αξιολογηθεί το σχέδιο και κατά πόσο αυτά τα οποία έχουμε σκεφτεί είναι υλοποιήσιμα.

- **Στάδιο προγραμματιστικής προετοιμασίας (Pre-Programming Phase)**

Αν το παιχνίδι έχει φτάσει μέχρι εδώ είμαστε πλέον έτοιμοι να ξεκινήσουμε την τεκμηρίωση του παιχνιδιού σαν σύνολο. Αρχικά θα πρέπει να αποτυπώσουμε γραπτώς το σύνολο των αποτελεσμάτων της μελέτης από την προηγούμενη φάση και να ορίσουμε πλέον από τεχνικής πλευράς το I/O structure και την αλγοριθμική δομή του παιχνιδιού. Ο στόχος αυτής της τεκμηρίωσης είναι να εντοπισθούν έγκαιρα πιθανά προβλήματα και δυσκολίες στηριζόμενοι σε μίας μορφής ψευδοκώδικα και όχι σε τεχνικές εκτιμήσεις. Τέλος γίνεται σύγκριση αυτών των εγγράφων με τα αρχικά που περιγράφουν την προγραμματιστική δομή του παιχνιδιού, έτσι ώστε να προσαρμοστούν τα δεύτερα σε περίπτωση που κριθεί απαραίτητο.

- **Στάδιο προγραμματισμού (Programming Phase)**

Σε αυτή την φάση αποτυπώνονται οι ιδέες του παιχνιδιού σε πραγματικό κώδικα. Είναι πολύ σημαντικό όλοι οι αλγόριθμοι που θα υλοποιηθούν να δουλεύουν τόσο σωστά όσο και αποδοτικά, προκειμένου το πρόγραμμά μας να έχει όσο το δυνατόν χαμηλότερες απαιτήσεις συστήματος. Οι αλγόριθμοι και οι τεχνικές υλοποίησης ενός video game περιγράφονται παρακάτω αναλυτικότερα.

- **Στάδιο Ελέγχου (Testing Phase)**

Σε ιδανική περίπτωση, το στάδιο ελέγχου είναι μια διαδικασία που παράγει πληροφορίες που χρησιμοποιούνται για την βελτίωση του παιχνιδιού. Στην πράξη όμως, αποκαλύπτει προβλήματα τόσο στον θεμελιώδη σχεδιασμό όσο και στην προγραμματιστική υλοποίηση του παιχνιδιού, τα οποία απαιτούν σημαντικές προσπάθειες για τη διόρθωση τους. Έτσι η φάση του ελέγχου είναι συχνά συνυφασμένη με τον προγραμματισμό και είναι γνωστή και ως εντοπισμός και διόρθωση σφαλμάτων (debugging).

Μερικές φορές το στάδιο ελέγχου αποκαλύπτει ότι το παιχνίδι έχει πολύ σοβαρά ελαττώματα για να σωθεί. Η ύπαρξη ενός τέτοιου "θανατηφόρου" ελαττώματος μπορεί να σημαίνει αναθεώρηση και σχεδίαση όλου του παιχνιδιού από την αρχή. Ένα "θανατηφόρο" ελάττωμα προκύπτει από μια θεμελιώδη σύγκρουση μεταξύ δύο σημαντικών στοιχείων του παιχνιδιού, η ασυμβατότητα των οποίων δεν είχε προβλεφθεί και η τροποποίηση μετά το στάδιο υλοποίησης του παιχνιδιού να μην είναι εφικτή λόγω του ότι το πρόγραμμά μας είναι σχεδιασμένο ανεπαρκώς.

Μια από τις τελευταίες εργασίες που πρέπει να εκτελεστεί πριν από την απελευθέρωση της τελικής έκδοσης του παιχνιδιού είναι η προετοιμασία του εγχειρίδιου του παιχνιδιού (manual). Η δημιουργία του εγχειριδίου γίνεται παράλληλα με το στάδιο ελέγχου καθώς σε αυτό το στάδιο εξετάζονται όλα τα επιμέρους κομμάτια του παιχνιδιού, κάτι που μας βοηθάει στο να μην παραλείψουμε κάποια σημαντική πληροφορία από το εγχειρίδιο. Μεγάλο μέρος των στατικών πληροφοριών που σχετίζονται με το παιχνίδι πρέπει να παρουσιάζονται σε αυτό το εγχειρίδιο. Το εγχειρίδιο είναι επίσης το κατάλληλο μέρος για να εισαχθούν φανταστικά στοιχεία, όπως εικόνες και ιστορίες. Τέλος, ένα καλογραμμένο εγχειρίδιο ξεκαθαρίζει πολλές από τις παρανοήσεις που συχνά προκύπτουν κατά τη διάρκεια που ο χρήστης παίζει το παιχνίδι.

2.7 Αναφορά στα Game Engines

Ένα game engine 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, ένα physics engine ή αλλίως μηχανή εντοπισμού συγκρούσεων (collision detection, καθώς και collision response), ήχο, scripting, animation, τεχνητή νοημοσύνη, δικτύωση, streaming, διαχείριση μνήμης και νήματα (threading). Έτσι η διαδικασία της ανάπτυξης ενός παιχνιδιού εξοικονομείται αν σκεφτούμε ότι μία τέτοια μηχανή επαναχρησιμοποιείται για να δημιουργηθούν και άλλα παιχνίδια.

Οι μηχανές παιχνιδιών παρέχουν και πακέτα οπτικών εργαλείων ανάπτυξης εκτός από ένα επαναχρησιμοποιήσιμο λογισμικό. Αυτά τα εργαλεία γενικά παρέχονται σε ένα ολοκληρωμένο περιβάλλον ανάπτυξης ώστε να καθιστούν εφικτή την απλή και γρήγορη ανάπτυξη παιχνιδιών. Αυτές οι μηχανές παιχνιδιών συχνά καλούνται "game middleware" επειδή, όπως με την εμπορική έννοια του όρου, παρέχουν μια ευέλικτη και επαναχρησιμοποιήσιμη πλατφόρμα λογισμικού η οποία παρέχει όλη την κεντρική λειτουργικότητα που απαιτείται για την ανάπτυξη μιας εφαρμογής παιχνιδιού ενώ ταυτόχρονα μειώνει τα κόστη, τις πολυπλοκότητες και το χρόνο μέχρι το τελικό προϊόν να βγει στην αγορά (time-to-market), παράγοντες που θεωρούνται πολύ σημαντικοί στην βιομηχανία των video games.

Όπως οι περισσότερες λύσεις middleware, οι μηχανές παιχνιδιών είναι συνήθως ανεξάρτητες από τις διάφορες πλατφόρμες, επιτρέποντας έτσι στο παιχνίδι να τρέχει σε διάφορες μηχανές, συμπεριλαμβανομένων των κονσόλων παιχνιδιών και των προσωπικών υπολογιστών, με ελάχιστες έως καθόλου αλλαγές στον πηγαίο κώδικα του παιχνιδιού. Συχνά μία middleware engine σχεδιάζεται βασιζόμενη σε αρχιτεκτονικές οι οποίες είναι συμβατές με συγκεκριμένα συστήματα hardware που μπορεί στο μέλλον να αντικατασταθούν ή να επεκταθούν με πιο εξειδικευμένα (και συχνά πιο ακριβά) στοιχεία middleware όπως το Havok για φυσική, το FMOD για ήχο, ή το Scaleform για UI και Βίντεο. Μερικές μηχανές παιχνιδιών όπως η RenderWare είναι σχεδιασμένες ως μια σειρά "χαλαρών συνδεδεμένων στοιχείων" middleware τα οποία μπορούν να συνδυαστούν κατά βούληση για τη δημιουργία μιας πιο εξειδικευμένης μηχανής, σε αντίθεση με την πιο κοινή προσέγγιση της επέκτασης ή της προσαρμογής μιας ευέλικτης game engine. Με οποιοδήποτε τρόπο και αν επιτυγχάνεται η επεκτασιμότητα, παραμένει μια υψηλή προτεραιότητα στις μηχανές παιχνιδιών λόγω της ευρείας ποικιλίας χρήσεων στις οποίες αυτές χρησιμοποιούνται. Παρά του ονόματος τους, οι μηχανές παιχνιδιών συχνά χρησιμοποιούνται και σε άλλου είδους real time διαδραστικές εφαρμογές με γραφικές απαιτήσεις όπως

παρουσιάσεις μάρκετινγκ, αρχιτεκτονικές αναπαραστάσεις, εκπαιδευτικές εξομοιώσεις και περιβάλλοντα μοντελοποίησης.

Μερικές μηχανές παιχνιδιών έχουν μόνο δυνατότητες real time 3D rendering, αντί για μία ευρεία γκάμα λειτουργιών που απαιτούνται για την ολοκληρωμένη κατασκευή παιχνιδιών. Αυτές οι μηχανές αφήνουν τον δημιουργό του παιχνιδιού είτε να υλοποιήσει μόνος του τις λειτουργίες αυτές, είτε να τις συνθέσει από άλλες middleware engines. Αυτού του τύπου οι μηχανές είναι γενικότερα γνωστές ως «graphic engines», «rendering engine» ή «3D engine» αντί του γενικότερου ονόματος «game engine». Ωστόσο, αυτή η ορολογία πολλές φορές χρησιμοποιείται λανθασμένα καθώς πολλά engines 3D παιχνιδιών, που υποστηρίζουν μία πληθώρα λειτουργιών, αναφέρονται απλά ως «3D engines». Μερικά παραδείγματα τέτοιων graphic engine είναι τα εξής: RealmForge, Truevision3D, OGRE, Crystal Space, Genesis3D, Irrlicht και JMonkey Engine. Οι μοντέρνες game engines ή graphic γενικά παρέχουν ένα scene graph, το οποίο είναι κάτι σαν αντικειμενοστραφής αναπαράσταση του 3D κόσμου του παιχνιδιού, η οποία συχνά απλοποιεί τον σχεδιασμό του παιχνιδιού και μπορεί να χρησιμοποιηθεί για πιο αποδοτικό rendering τεράστιων εικονικών κόσμων.

Πιο συχνά, οι 3D μηχανές ή τα συστήματα rendering στις μηχανές παιχνιδιών κατασκευάζονται πάνω σε ένα API γραφικών, όπως τα Direct3D ή OpenGL, το οποίο δίνει μία ανεξαρτησία του λογισμικού από τις GPU ή γνωστές και ως κάρτες γραφικών. Βιβλιοθήκες χαμηλού επιπέδου όπως οι DirectX, SDL και OpenAL χρησιμοποιούνται συχνά σε παιχνίδια καθώς παρέχουν ανεξαρτησία από υλικό σε υλικό ενός υπολογιστή, όπως είναι οι συσκευές εισόδου, οι κάρτες δικτύου και οι κάρτες ήχου. Εκτός από τα 3D accelerated graphics πολλές φορές χρησιμοποιούνται και software renderers. Το software rendering χρησιμοποιείται σε κάποια εργαλεία μοντελοποίησης ή για rendering φωτογραφιών και εικόνων, όταν δίνεται προτεραιότητα στην οπτική ακρίβεια σε σχέση με την real time επίδοση (FPS) ή όταν το υλικό του υπολογιστή δεν έχει πλήρη τις ελάχιστες απαιτήσεις συστήματος ή δεν υποστηρίζει τεχνολογίες όπως shader 2.0 ή Direct3D 10.

Με την ανάπτυξη των 3D accelerated graphics και του physics proccessing, διάφορα physics API όπως το PAL και οι επεκτάσεις του (COLLADA), μια μηχανή εναλλακτικής απεικόνισης 3D στοιχείων, έγιναν διαθέσιμα προς χρήση παρέχοντας έναν αφαιρετικό τρόπο ανάπτυξης λογισμικού όσον αφορά το physics processing, ανεξαρτήτως του middleware engine και της πλατφόρμας που χρησιμοποιείται.

Πριν από τις μηχανές παιχνιδιών, τα παιχνίδια τυπικά υλοποιούνταν ως μοναδικές οντότητες, για παράδειγμα ένα παιχνίδι για το Atari 2600, έπρεπε να σχεδιαστεί από το μηδέν για να κάνει βέλτιστη χρήση του hardware, κάτι το οποίο σήμερα επιτυγχάνεται από την κεντρική ρουτίνα εμφάνισης που λέγεται retro core. Μεταγενέστερες πλατφόρμες είχαν περισσότερες δυνατότητες, αλλά ακόμα και όταν τα γραφικά δεν είχαν υψηλές απαιτήσεις, οι περιορισμοί σε μνήμη συνήθως οδηγούσαν σε αποτυχία τις προσπάθειες για τη δημιουργία ενός παιχνιδιού που απαιτούσε την επεξεργασία περισσότερων δεδομένων. Ακόμα και στις πιο φιλικές

πλατφόρμες από άποψη σχεδίασης, λίγα ήταν τα παιχνίδια που μπορούσαν να υποστηρίξουν, πέραν εκείνου του παιχνιδιού για το οποίο είχαν αρχικά φτιαχτεί. Η ανάπτυξη του υλικού για arcade παιχνίδια, ο ακρογωνιαίος λίθος της αγοράς εκείνα τα χρόνια, σήμαινε ότι το μεγαλύτερο μέρος του κώδικα θα έπρεπε να πεταχτεί αργότερα, αφού τα επόμενα παιχνίδια θα χρησιμοποιούσαν τελείως διαφορετικές τεχνικές σχεδίασης, οι οποίες εκμεταλλευόντουσαν επιπλέον πόρους. Τοιουτοτρόπως οι περισσότερες σχεδιάσεις παιχνιδιών τη δεκαετία του 1980, απαρτιζόντουσαν από ένα hard coded σύνολο, με μια μικρή ποσότητα δεδομένων και απλοϊκών γραφικών.

Η πρώτη γενιά των graphic engines ή renderers, καθώς και οι πρόγονοι αυτών, κυριαρχούταν από τρία λογισμικά προγράμματα: το BRender από την Argonaut Software, το Renderware από την Criterion Software Limited και το Reality Lab της RenderMorphics. Το Reality Lab ήταν το ταχύτερη από τα τρία και χαρακτηρίστηκε ως μία από τις πιο επιθετικές κινήσεις της Microsoft εκείνη την εποχή. Η ομάδα της RenderMorphics, Servan Keondjian, Kate Seekings και Doug Rabson ακολούθως προσχώρησαν στο εγχείρημα της Microsoft το οποίο μετέτρεψε το Reality Lab στο Direct3D. Το Renderware αργότερα αγοράστηκε από την EA, αλλά τελικά η εταιρεία γίγαντας των video games σταμάτησε να το υποστηρίζει και να το αναπτύσσει.

Ο όρος «game engine» ήρθε στην επιφάνεια στα μέσα της δεκαετίας του 1990, με αφορμή τα 3D παιχνίδια της εποχής και κυρίως αναφερόμαστε στα παιχνίδια σκόπευσης πρώτου προσώπου (first person shooting). Η δημοτικότητα των παιχνιδιών Doom και Quake ήταν τόσο μεγάλη εκείνη την εποχή, που αντί να κατασκευάζονται παιχνίδια από μηδενική βάση, οι σχεδιαστές αγόραζαν τις άδειες χρήσης των παραπάνω τίτλων για τα δομικά στοιχεία του παιχνιδιού τους και σχεδίασαν αργότερα τα δικά τους γραφικά, χαρακτήρες, όπλα και επίπεδα Ο διαχωρισμός τέτοιων ειδικών λειτουργιών και δεδομένων από βασικές έννοιες, όπως ο έλεγχος σύγκρουσης και η απόδοση 3D οντοτήτων στο χώρο, σήμαινε ότι οι ομάδες υλοποίησης βιντεοπαιχνιδιών μπορούσαν πλέον να μεγαλώσουν και να εξειδικευτούν. Μεταγενέστερα παιχνίδια, όπως το Quake III Arena και το Unreal Tournament του 1998 της Epic Games σχεδιάζονταν με αυτή την προσέγγιση στο μυαλό, με το engine και το content αναπτυγμένα ξεχωριστά. Η πρακτική της αδειοδότησης τέτοιας τεχνολογίας έχει αποδειχθεί ότι είναι μια πολύ καλή και χρήσιμη πρακτική ανεύρεσης βοηθητικής πηγής εσόδων για τις εταιρείες δημιουργίας βιντεοπαιχνιδιών, καθώς μια τέτοια άδεια για τη χρήση μίας υψηλής ποιότητας εμπορικής game engine μπορεί να κυμανθεί από $10,000 μέχρι μερικά εκατομμύρια δολάρια. Επίσης, τα επαναχρησιμοποιήσιμα game engines κάνουν την ανάπτυξη παιχνιδιών σειράς γρηγορότερη και ευκολότερη, το οποίο είναι ένα πολύ σημαντικό πλεονέκτημα στην ανταγωνιστική βιομηχανία βιντεοπαιχνιδιών.

Ένα σύγχρονο game engine είναι ίσως μία από τις πιο πολύπλοκες εφαρμογές που μπορεί να υλοποιήσει κανείς και αυτό γιατί συχνά διαθέτουν τελειοποιημένα συστήματα αλληλεπίδρασης, προκειμένου να εγγυώνται στον χρήστη ακριβή

χειρισμό και μοναδικές εμπειρίες. Η συνεχής εξέλιξη των game engines έχει δημιουργήσει ένα ισχυρό διαχωρισμό ανάμεσα στο rendering, το scripting, το graphic design και το level design. Είναι πλέον σύνηθες φαινόμενο, για παράδειγμα μια ομάδα ανάπτυξης παιχνιδιών να έχει πολλές φορές περισσότερους σχεδιαστές καλών τεχνών απ' ότι προγραμματιστές. Τα παιχνίδια σκόπευσης πρώτου προσώπου παραμένουν οι κυρίαρχοι χρήστες των game engines από τρίτους, αν και πλέον χρησιμοποιούνται και σε άλλα είδη παιχνιδιών. Όπως για παράδειγμα το RPG The Elder Scrolls III: Morrowind και το MMORPG Dark Age of Camelot, τα οποία βασίζονται στη Gamebryo engine και το MMORPG Lineage II βασίζεται στην Unreal Engine. Ακόμη οι μηχανές παιχνιδιών χρησιμοποιήθηκαν και για παιχνίδια που αρχικά ήταν ανεπτυγμένα για οικιακές κονσόλες όπως για παράδειγμα η μηχανή RenderWare που χρησιμοποιήθηκε στις σειρές παιχνιδιών Grand Theft Auto και Burnout.

Ο προγραμματισμός με νήματα γίνεται όλο και πιο σημαντικός λόγω των μοντέρνων πολυπύρηνων συστημάτων (π.χ. ο Cell της Sony και των αυξημένων απαιτήσεων σε ρεαλισμό. Συνήθως ξεχωριστά νήματα αναλαμβάνουν το rendering, το streaming, τον ήχο και τα physics. Τα αγωνιστικά παιχνίδια βρίσκονται στην πρώτη γραμμή του προγραμματισμού με νήματα, με τη Physics engine να τρέχει ένα ξεχωριστό νήμα στον πυρήνα σε σχέση με άλλες διεργασίες όπως για παράδειγμα το rendering και σχετικές διεργασίες που απαιτούν μόνο ενημέρωση στα 30-60 Hz και επομένως μπορούν να μοιράζονται τον πυρήνα λόγω χαμηλών απαιτήσεων. Για παράδειγμα, το Need For Speed στο Playstation τρέχει τα Physics του στα 100 Hz σε σχέση με το Forza Motorsport 2 που τρέχει τη Φυσική του στα 360 Hz. Αν και ο όρος αρχικά χρησιμοποιήθηκε στη δεκαετία του 1990, υπάρχουν και κάποια προγενέστερα συστήματα στη δεκαετία του 1980 τα οποία επίσης θεωρούνται μηχανές παιχνιδιών, όπως τα συστήματα AGI και SCI της Sierra, το σύστημα SCUMM της LucasArts και η μηχανή Freescape της Incentive Software. Ωστόσο, αντίθετα με τις μοντέρνες μηχανές παιχνιδιών, αυτά τα συστήματα δεν χρησιμοποιήθηκαν ποτέ σε προϊόντα τρίτων.

Καθώς η τεχνολογία των μηχανών παιχνιδιών ωριμάζει και γίνεται πιο φιλική προς το χρήστη, η εφαρμογή των μηχανών παιχνιδιών έχει διευρυνθεί σε μέγεθος και πλέον χρησιμοποιείται και σε πιο σοβαρά project, όπως είναι η οπτικοποίηση αντικειμένων, η εκπαίδευση, η ιατρική και η στρατιωτικής εξομοίωση. Για να διευκολύνουν αυτή την προσβασιμότητα, νέες πλατφόρμες υλικού υποστηρίζονται σήμερα από τις σύγχρονες μηχανές παιχνιδιών, συμπεριλαμβανομένου των κινητών τηλεφώνων και των internet browsers. Επίσης, περισσότερες μηχανές παιχνιδιών κατασκευάζονται πάνω σε Υψηλού επιπέδου γλώσσες προγραμματισμού όπως η Java και η C#/.NET, όπως η TorqueX, η Blade3D και Visual3D.NET. Καθώς τα περισσότερα από τα παιχνίδια υψηλών γραφικών περιορίζονται ως επί το πλείστον από την GPU, οι ενδεχόμενες καθυστερήσεις από τις γλώσσες προγραμματισμού υψηλού επιπέδου γίνονται αμελητέες, ενώ τα οφέλη παραγωγικότητας που προσφέρονται από αυτές τις γλώσσες στους δημιουργούς μηχανών παιχνιδιών είναι

πολύ περισσότερα. Αυτές οι πρόσφατες τάσεις προωθούνται κυρίως από εταιρείες όπως είναι η Microsoft, προκειμένου να μπορούν να υποστηρίξουν την ανάπτυξη παιχνιδιών ανεξάρτητα από την πλατφόρμα για την οποία προορίζονται, όπως συμβαίνει με το Xbox360 και το Zune, τα οποία χρησιμοποιούν το .NET Framework και το XNA για το rendering των γραφικών και του ήχου. Έτσι γίνεται ευκολότερη και φθηνότερη από ποτέ η δημιουργία των game engines για πλατφόρμες οι οποίες υποστηρίζουν και μοιράζονται τα ίδια API's ακόμα και όταν αυτά τρέχουν σε διαφορετικό hardware.

Κάποιες εταιρείες εξειδικεύονται στην ανάπτυξη πακέτων λογισμικού middleware που αναφέραμε ποηγουμένως. Οι δημιουργοί των middleware's αποπειρώνται να "ξαναανακαλύψουν τον τροχό" με την ανάπτυξη πακέτων λογισμικού οι οποίες περιλαμβάνουν ευρείας γκάμας εργαλεία και πολλών διαφορετικών στοιχείων που ένας δημιουργός παιχνιδιού μπορεί να χρειαστεί για να κατασκευάσει ένα παιχνίδι. Τα περισσότερα προγράμματα middleware παρέχουν δομές που κάνουν ευκολότερη την ανάπτυξη, όπως λειτουργίες γραφικών, ήχου, Physics και τεχνητής νοημοσύνης. Η Gamebryo και η RenderWare μερικά από τα πιο διαδεδομένα middleware λογισμικά. Υπάρκουν μερικά middleware που μπορούν να κάνουν μόνο ένα πολύ συγκεκριμένο πράγμα, αλλά το κάνουν τόσο καλά και τόσο αποδοτικά σε σχέση με τις μηχανές γενικού σκοπού, γι αυτό και προτιμούνται. Για παράδειγμα, η SpeedTree χρησιμοποιήθηκε για να κάνει ρεαλιστικό rendering στα δέντρα και στη βλάστηση στο παιχνίδι ρόλων The Elder Scrolls IV: Oblivion. Τα τέσσερα πιο ευρέως χρησιμοποιημένα πακέτα middleware, παρέχουν υποσυστήματα λειτουργικότητας όπως το Bink της RAD Game Tools, το Firelight FMOD, το Havok, και το Scaleform GFx. Η RAD Game Tools αναπτύσσει το Bink για βασικό video renderer, μαζί με το Miles audio και το Granny 3D rendering. Το Firelight FMOD είναι μια χαμηλού κόστους βιβλιοθήκη ήχου και παρόμοιων εργαλείων. Το Havok παρέχει ένα εύρωστο σύστημα εξομοίωσης, μαζί με μια σουίτα για animation και τεχνητή νοημοσύνη. Το Scaleform παρέχει GFx για υψηλής απόδοσης Flash UI, μαζί με εργαλεία αναπαραγωγής υψηλής ποιότητας βίντεο και έναν πρόσθετο Επεξεργαστή Μεθόδου Εισόδου (IME) για την υποστήριξη chat μέσα στο παιχνίδι. Μερικά middleware περιέχουν πλήρη πηγαίο κώδικα, άλλα απλά παρέχουν ένα API για τη βιβλιοθήκη του μεταγλωττιστή. Επίσης σε κάποια προγράμματα middleware υπάρχει η δυνατότητα αδειοδότησης με τον πλήρη πηγαίο κώδικα πληρώνοντας βέβαια πάντα κάτι παραπάνω.

Ένα πολύ γνωστό υποσύνολο των μηχανών παιχνιδιών είναι οι μηχανές παιχνιδιών των 3D παιχνιδιών σκόπευσης πρώτου προσώπου (FPS). Η ανάπτυξη των οποίων σε όρους οπτικής ποιότητας είναι ρηξικέλευθη και γίνεται σε ανθρώπινη κλίμακα. Ενώ οι εξομοιωτές πτήσης και οδήγησης και τα παιχνίδια στρατηγικής πραγματικού χρόνου (RTS) παρέχουν υψηλό και συνεχώς αυξανόμενο ρεαλισμό, τα παιχνίδια σκόπευσης πρώτου προσώπου βρίσκονται παρόλα αυτά στην πρώτη γραμμή των απαιτήσεων σε γραφικά. Η ανάπτυξη των μηχανών γραφικών FPS οι οποίες εμφανίζονται στα παιχνίδια μπορεί να χαρακτηριστεί από μια σταθερή αύξηση

των τεχνολογιών και των καινοτομιών. Προσπάθειες στο να οριστούν διακριτές γενιές, οδηγούν σε αυθαίρετες επιλογές του τι αποτελεί μια βαριά τροποποιημένη έκδοση μια 'παλιάς μηχανής' και του τι είναι μια καινούργια μηχανή. Η κατάταξη είναι πολύπλοκη καθώς οι μηχανές παιχνιδιών συνδυάζουν παλιές και νέες τεχνολογίες. Χαρακτηριστικά που θεωρούνται προχωρημένα σε ένα νέο παιχνίδι τον ένα χρόνο, γίνονται αναμενόμενα και αναγκαία τον επόμενο χρόνο. Η νόρμα είναι πλέον παιχνίδια με μία μίξη χαρακτηριστικών νέας και παλαιότερης γενιάς. Για παράδειγμα το Jurassic Park: Trespasser (1998) εισήγαγε την Φυσική στα παιχνίδια FPS, αλλά αυτό δεν έγινε γνωστό μέχρι το 2002. Το Red Faction (2001) χαρακτηριζόταν από πλήρως καταστρέψιμο περιβάλλον, τοίχους και έδαφος, κάτι που ακόμα και σημερινές μηχανές τόσα χρόνια μετά δεν διαθέτουν. Επίσης το Battlezone (1998) και το Battlezone II: Combat Commander (1999) ήταν τα πρώτα παιχνίδια που πρόσθεσαν μάχη βασισμένη σε οχήματα στα παιχνίδια σκοπεύσεως πρωτου προσώπου, κάτι το οποίο δεν ήταν δημοφιλές μέχρι πρόσφατα.

2.8 Αναφορά στους Level Editors

Όταν άρχισαν να πρωτοεμφανίζονται τα βιντεοπαιχνίδια, συνήθως ένα άτομο ήταν υπεύθυνο για την κατασκευή όλου του παιχνιδιού, δηλαδή να δημιουργήσει τόσο τον κώδικα όσο και τα γραφικά που θα υποστήριζε και αυτό γιατί δεν είχε προκύψει ακόμη η ανάγκη για μια ειδικότητα η οποία θα ασχολιόταν αποκλειστικά και μόνο με το level design του παιχνιδιού. Έτσι λοιπόν στα πρώτα παιχνίδια καθώς το σενάριο εξελισσόταν απλά το παιχνίδι γινόταν όλο και πιο δύσκολο, εμφανίζονταν παραδείγματος χάριν πιο πολλά εμπόδια ή αυξανόταν η ταχύτητα των αντικειμένων. Τα πρώτα παιχνίδια που χρειάστηκαν πολύ μεγάλο χρονικό διάστημα για να σχεδιαστούν ήταν τα λεγόμενα text-based. Εκεί λόγω τις αφθονίας των επιλογών, δινόταν η δυνατότητα στους χρηστές να σχεδιάσουν οι ίδιοι κάποια επίπεδα, όπως καινούργιους δρόμους, καινούργια δωμάτια ακόμη και καινούργιο εξοπλισμό έτσι ώστε να ενσωματωθούν στο ήδη υπάρχων σενάριο του εκάστου παιχνιδιού. Έτσι με την πάροδο του χρόνου, αναπτύχτηκε ένας μεγάλος κλάδος που ασχολείται αποκλειστικά με αυτό το κομμάτι των παιχνιδιών και ονομάζεται level design.

Ο level editor είναι ένα εργαλείο software, το οποίο χρησιμοποιείται για να δημιουργεί χάρτες ή αλλιώς επίπεδα μέσα σε ένα παιχνίδι. Για να γίνουμε πιο κατανοητοί σε άτομα τα οποία δεν έχουν ασχοληθεί με την βιομηχανία των παιχνιδιών θα εξηγήσουμε λίγα πράγματα γύρω από την φιλοσοφία που ακολουθούν σχεδόν όλα τα παιχνίδια. Όταν τελικά αποφασίσουμε ποιο θα είναι το σενάριο θα πρέπει να δημιουργήσουμε και τον κόσμο στον οποίο θα εκτυλίσσετε το παιχνίδι μας.

Εδώ έρχεται και αναλάβει ρόλο ο level editor. Με αυτό το εργαλείο μπορούμε να κατασκευάσουμε χάρτες του παιχνιδιού, τους οποίους ονομάζουμε "πίστες" ή αλλιώς επίπεδα. Επίσης μπορούμε να δημιουργήσουμε ολόκληρα σενάρια μέσα στα οποία ο παίχτης θα πρέπει να πραγματοποιήσει κάποιες ενέργειες και φυσικά όταν υπάρχει σενάριο κάποιος θα πρέπει να είναι και ο σκηνοθέτης, δηλαδή να ορίσει ποιες γωνίες λήψεις θέλει, πόσο κοντά ή μακριά θα φαίνονται τα αντικείμενα, πως θα πρέπει να γίνονται οι εναλλαγές της κάμερας έτσι ώστε να φαίνεται αισθητικά ένα πιο ωραίο αποτέλεσμα. Εδώ έρχεται ο game level editor, ο οποίος δίνει δυνατότητες χειρισμού της κάμερας κάνοντας τον παίχτη να νιώθει ότι είναι μέρος ενός σεναρίου που εκτυλίσσεται, καθώς παίζει το παιχνίδι. Υπάρχουν παράλληλα και πολλές άλλες λειτουργίες που μπορούμε να ενσωματώσουμε στον editor μας που μπορούν να κάνουν ποιο εξειδικευμένες εργασίες όπως να δημιουργούμε καινούργιους αντιπάλους και αντικείμενα και να τους δίνουμε μοναδικά χαρακτηριστικά στο καθένα, όπως το σχήμα που θα έχουν, την δυσκολία που χρειάζεται ο παίχτης να σκοτώσει έναν αντίπαλο κτλ.. Ακόμη μπορούμε να αλλάζουμε το background του παιχνιδιού προσθέτοντας καινούργιες εικόνες. Για να κάνουμε το παιχνίδι μας ακόμη ποιο εντυπωσιακό, έχουμε την δυνατότητα να βάλουμε και Parallax, το οποίο μας δίνει την αίσθηση του βάθους αλλά και της ταχύτητας. Ακόμη και η μουσική του παιχνιδιού είναι εύκολο να αλλάξει και κάθε φορά να μπορεί ο δημιουργός να δώσει την ένταση και τον τόνο που χρειάζεται. Εάν πάμε σε ποιο σύνθετες μηχανές παιχνιδιών, μπορούμε να κατασκευάσουμε level editors που δημιουργούν καινούργιους κόσμους, με ιδιαίτερα εντυπωσιακές λειτουργίες. Για παράδειγμα, σε παιχνίδια που είναι 3d και διαδραματίζονται σε κάποιο εικονικό κόσμο, έχουμε την δυνατότητα να προσθέσουμε καινούργια αντικείμενα, να κτίσουμε έναν καινούργιο κόσμο εμείς οι ίδιοι όπως σπίτια, δάση, ποτάμια, βουνά. Οι δυνατότητες είναι απίστευτες και εξαρτώνται από τις απαιτήσεις του εκάστοτε παιχνιδιού.

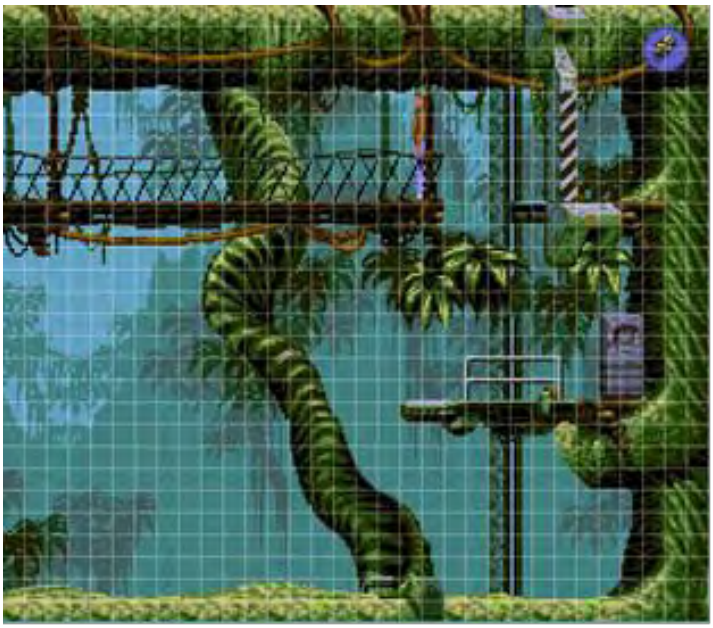
Ο ρόλος που επιτελεί ένας game level designer είναι ιδιαίτερα σημαντικός γιατί γεφυρώνει το χάσμα που συνήθως υπάρχει μεταξύ ενός γραφίστα και ενός προγραμματιστή. Ένα παιχνίδι το οποίο θέλουμε να έχει κάποιο γραφικό περιβάλλον, απαιτεί τις περισσότερες φορές και την συμβολή ενός γραφίστα ή κάποιου καλλιτέχνη ο οποίος θα επιμεληθεί τα γραφικά-animation του παιχνιδιού. Έτσι δίνει χρώμα αλλά και μια δόση αληθοφάνειας και κάνει το σενάριο πιο ευχάριστο και πιο καλαίσθητο. Γι αυτό το πρόγραμμα μας θα πρέπει να δίνει την δυνατότητα στον γραφίστα να αποτυπώσει την ιδέα του με άπλες εντολές ή ακόμη καλυτέρα με κάποια κουμπιά. Φυσικά αναλόγως τις επιλογές που θέλουμε να του δώσουμε, μπορούμε να προσθέσουμε λειτουργίες, γι αυτό καλό θα είναι να υπάρχει μια επικοινωνία για το τι ακριβώς θα ήταν σκόπιμο να ενσωματωθεί πάνω στο πρόγραμμα.

Υπάρχουν αρκετοί τρόποι να δημιουργηθεί ένας editor, ανάλογα με τις δυνατότητες που θέλουμε να του δώσουμε. Θα κάνουμε παρακάτω μια ανάλυση σε μεθόδους που υπάρχουν και τι ακριβώς προσφέρει η καθεμιά. Να τονιστεί ότι εμείς θα ασχοληθούμε κυρίως με 2d πλατφόρμες παιχνιδιών.

Tile-based(pure)

Η κίνηση του χαρακτήρα είναι περιορισμένη σε «πλακάκια», έτσι ώστε να μην μπορεί να σταθεί στα μισά του δρόμου ανάμεσα σε δύο πλακίδια. Κινούμενα σχέδια μπορούν να χρησιμοποιηθούν για να δημιουργήσουν την ψευδαίσθηση της ομαλής κίνησης, αλλά όσον αφορά την λογική του παιχνιδιού, ο παίκτης πρέπει να είναι πάνω από ένα συγκεκριμένο «πλακίδιο». Αυτός είναι ο ευκολότερος τρόπος για να εφαρμόσει ένα παιχνίδι πλατφόρμας, αλλά επιβάλλει μεγάλους περιορισμούς σχετικά με τον έλεγχο του χαρακτήρα, καθιστώντας το ακατάλληλο για παιχνίδια που βασίζονται στην δράση. Είναι όμως δημοφιλής για παιχνίδια τύπου πάζλ .

Στην εικόνα 1 βλέπουμε μια απλή υλοποίηση του γραφικού περιβάλλοντας τύπου πλακιδίων.



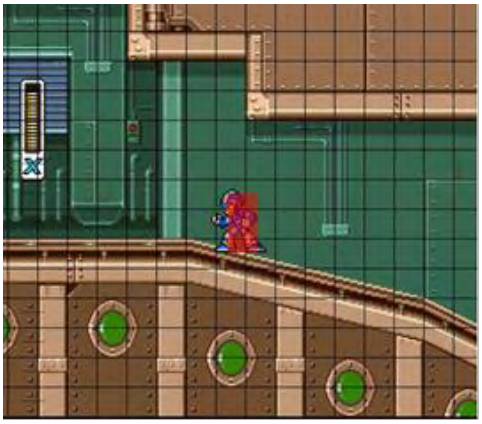
εικόνα 1

Πώς λειτουργεί:

Ο χάρτης είναι ένα πλέγμα πλακιδίων που στο καθένα από αυτά αποθηκεύονται πληροφορίες, όπως το αν είναι ένα εμπόδιο ή όχι, ποια εικόνα πρέπει να χρησιμοποιηθεί, τι είδους ήχος θα παραχθεί εάν γίνει ένα βήμα πάνω σε αυτά, και ούτω καθεξής. Ο παίκτης ή άλλοι χαρακτήρες αντιπροσωπεύονται από ένα σύνολο ενός ή περισσοτέρων πλακιδίων που κινούνται μαζί. Πλεονεκτήματα αυτού του συστήματος περιλαμβάνουν την απλότητα και την ακρίβεια. Δεδομένου ότι τα παιχνίδια είναι πιο ντετερμινιστικά, δυσλειτουργίες είναι λιγότερο πιθανόν να συμβούν, και η εμπειρία (gameplay) είναι πιο ελεγχόμενη.

Tile-based(smooth)

Η σύγκρουση εξακολουθεί να καθορίζεται από ένα tilemap, αλλά οι χαρακτήρες μπορούν να κυκλοφορούν ελεύθερα σε όλο τον κόσμο. Αυτή είναι η πιο κοινή μορφή της εφαρμογής platformers στις κονσόλες 8- bit και 16- bit, και παραμένει δημοφιλής σήμερα, διότι εξακολουθεί να είναι εύκολο να εφαρμοστεί και κάνει την επεξεργασία επίπεδου απλούστερη απ ότι πιο εξελιγμένες τεχνικές. Επιτρέπει, επίσης, ομαλά άλματα και είναι κατάλληλο για παιχνίδια δράσης που απαιτούν πιο σύνθετη κίνηση.



εικόνα 2

Πως λειτουργεί:

Οι πληροφορίες χάρτη είναι αποθηκευμένες κατά τον ίδιο τρόπο όπως και πριν, με την μόνη διαφορά ότι οι χαρακτήρες αλληλεπιδρούν με το φόντο. Με αυτήν την τεχνική έχουμε ένα καλύτερο gameplay και μεγαλύτερης ακρίβειας, κάνοντάς το πιο δίκαιο για τον παίχτη. Εδώ εμφανίζονται κλίσεις έτσι ώστε να κάνουν την πίστα ποιο δύσκολη, σκάλες ώστε να μπορεί ο παίχτης να ανεβεί σε ένα πιο ψηλό σημείο τις πίστας, ακόμη και σκαλοπάτια δίνοντας περισσότερη ένταση στο παιχνίδι. Τέλος μπορούμε να προσθέσουμε και άλλα χαρακτηριστικά στο σενάριο μας όπως μετακινούμε πλακίδια, ακόμη και ποιο σύνθετα, τα οποία όμως ξεφεύγουν από την μελέτη μας. Παράδειγμα τέτοιου τύπου παιχνιδιών βλέπουμε στις εικόνες 2 και 3.

εικόνα 3

Bitmask

Παρόμοια τεχνική με την προηγούμενη, αλλά αντί να χρησιμοποιούμε μεγάλα πλακάκια , μια εικόνα χρησιμοποιείται για τον προσδιορισμό σύγκρουσης για κάθε pixel. Αυτό επιτρέπει περισσότερες λεπτομέρειες, αλλά αυξάνει σημαντικά την πολυπλοκότητα , την χρήση μνήμης , και απαιτεί κάτι που μοιάζει με ένα πρόγραμμα επεξεργασίας εικόνας για να δημιουργήσουμε επίπεδα . Επίσης απαιτείται πολλές φορές να σχεδιάζονται μεγάλα κομμάτια γραφικών, τα οποία θα είναι ξεχωριστά για κάθε επίπεδο. Λόγω αυτών των ζητημάτων , αυτήν είναι μια σχετικά σπάνια τεχνική , αλλά μπορεί και παράγει υψηλότερα ποιοτικά αποτελέσματα από τις προαναφερθέντες λύσεις. Είναι επίσης κατάλληλη για δυναμικά περιβάλλοντα.

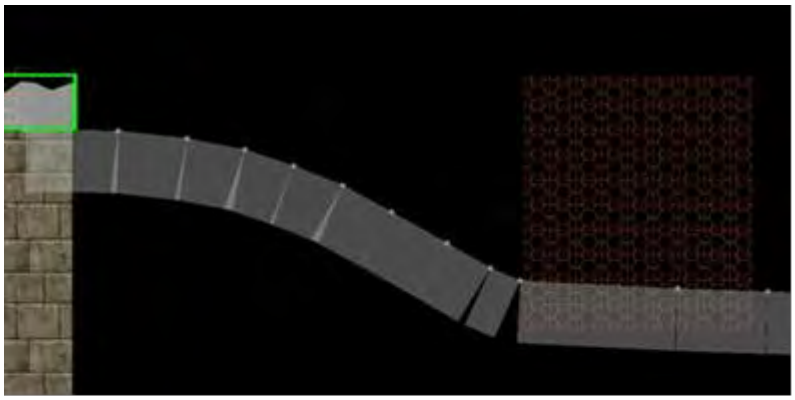
εικόνα 4

Πώς λειτουργεί:

Η βασική ιδέα είναι ίδια με τις προηγούμενες, απλά εδώ αλλάζει ο αλγόριθμος και γίνετε πιο περίπλοκος. Εδώ παίζουμε ιδιαίτερα με κλίσεις και αυτό κάνει τα πράγματα ακόμη πιο δύσκολα. Σε γενικές γραμμές , αυτή η τεχνική απαιτεί πολύ λεπτομέρεια και είναι εμφανώς λιγότερο σταθερή από τις προηγούμενες που βασίζονται στα πλακίδια. Συστήνετε μόνο για κάποιους που θέλουν να ενσωματώσουν λεπτομέρειες πάνω στο έδαφος του παιχνιδιού. Στην εικόνα 4 βλέπουμε ένα στιγμιότυπο του παιχνιδιού WORMS, το οποίο κάνει χρήση των παραπάνω λειτουργιών.

Vectorial

Η τεχνική αυτή χρησιμοποιεί vectorial δεδομένα (γραμμές ή πολύγωνα) για να καθοριστούν τα όρια των περιοχών σύγκρουσης . Πολύ δύσκολο να εφαρμοστεί σωστά, ωστόσο, είναι ολοένα και πιο δημοφιλής λόγω της αυξανόμενης παρουσίας των physics engines, όπως το Box2D , τα οποία είναι κατάλληλα για την εφαρμογή αυτής της τεχνικής . Παρέχει οφέλη παρόμοια με την τεχνική bitmask, αλλά χωρίς σημαντική κατανάλωση μνήμης , και χρησιμοποιώντας έναν πολύ διαφορετικό τρόπο από τα επίπεδα επεξεργασίας .



Στην εικόνα 5 βλέπουμε πως δομείται ένας vectorial level editor και πώς ορίζεται η λογική του παιχνιδιού.

Εικόνα 5

Και στην εικόνα 6 έχουμε την τελική μορφή, όπου έχουν προστεθεί χρώματα και τα διάφορα γραφικά.

Εικόνα 6

Πώς λειτουργεί:

Εδώ όπως είπαμε και παραπάνω, κύριο λόγο έχουν οι physics engines οι οποίες παρέχουν στον χρηστή ένα δυνατό εργαλείο για τον σχεδιασμό της πίστας. Βέβαια θέλει ιδιαίτερη προσοχή για να μην παρασυρθούμε και δημιουργήσουμε ένα περιβάλλον που θα είναι «πλαστικό» και θα στερείται φαντασίας και πολυμορφίας. Επίσης ο χρήστης μπορεί να ενσωματώσει περισσότερα χαρακτηριστικά, από την τριβή που θα έχει ο χαρακτήρας του όταν θα περιπατάει, μέχρι και την επιτάχυνση του. Λόγο της φυσικής που μπαίνει πλέον στο μοντέλο παιχνιδιού, η απόδοση γίνετε ακόμη πιο ρεαλιστική και τα χαρακτηριστικά της μετατρέπονται σε πλεονεκτήματα που απογειώνουν το gameplay.

Από τα παραπάνω σχηματίζουμε μια άποψη για τις τεχνικές που χρησιμοποιούνται για να δημιουργηθούν οι game level editors αλλά και το τι χαρακτηριστικά μπορούν αυτοί να ενσωματώνουν. Ο προγραμματιστής οφείλει κάνει το πρόγραμμα όσο πιο απλό γίνετε για να μπορεί ο γραφίστας ή κάποιος άλλος που θέλει να ασχοληθεί με το design του παιχνιδιού, να παρέμβει εύκολα. Έτσι μπορεί να σχεδιάσει όχι μόνο πίστες, αλλά και επιμέρους χαρακτηριστικά. Ιδιαίτερα τα τελευταία χρόνια παρατηρείται άτομα τα οποία έχουν σαν hobby το gaming, να δίνουν ιδιαίτερο ενδιαφέρον για τροποποιημένες εκδόσεις των παιχνιδιών, οι οποίες ενσωματώνουν καινούργιες πίστες, κρυμμένα χαρακτηριστικά, τα οποία ανεβάζουν

το παιχνίδι σε άλλο επίπεδο. Αυτοί λοιπόν οι χρήστες οι οποίοι θεωρούνται ερασιτέχνες, είναι σε θέση να δημιουργήσουν μέσω των game level editors δικές τους εκφάνσεις του παιχνιδιού, μερικές από τις οποίες γίνονται ιδιαίτερα δημοφιλής στις κοινότητες των gamers και μάλιστα πολλές φορές τα στάνταρ τους είναι ιδιαίτερα υψηλά. Γι αυτό και οι βιομηχανία παιχνιδιών επενδύει μεγάλα ποσά πάνω στην ανάπτυξη και υλοποίηση τέτοιων προγραμμάτων. Το level design είναι απαραίτητο για δύο ακόμα βασικούς λόγους - δίνει στους παίκτες έναν στόχο μέσα στο παιχνίδι και τους παρέχει μια ποιο απολαυστική εμπειρία παιχνιδιού. Ιδιαίτερα στα παιχνίδια που βασίζονται σε μια ιστορία, είναι καθοριστικό στο να υπάρχει μια σωστή και ομαλή ροη. Τα textures και ο ήχος συντελούν σε ένα άρτιο αποτέλεσμα.

Μια ευρεία ποικιλία εργαλείων μπορεί να χρησιμοποιηθεί από κάποιον για τον σχεδιασμό μίας πίστας. Αν και είναι πιο εύκολο να σχεδιάσουμε τα μοντέλα και τις υφές με εργαλεία γενικού σκοπού δημιουργίας πολυμέσων, τα παιχνίδια απαιτούν συνήθως τα δεδομένα να είναι σε μια μοναδική μορφή κατάλληλη για την μηχανή του παιχνιδιού (game engine). Για το σκοπό αυτό υπάρχουν ειδικοί compilers. Ενδεικτικά αναφέρουμε κάποια από αυτά τα εργαλεία γνωστών εταιριών όπως Bethesda Softworks's Construction Set, Valve's Hammer Editor, Epic's UnrealEd και UDK, Leadwerks 3D World Studio, BioWare's Aurora Toolset, id Software's Q3Radiant, Unity 3D κτλ.

3. Εισαγωγή στην C# και στο XNA

3.1 Εισαγωγή

Η ανάπτυξη ενός βιντεοπαιχνιδιού μπορεί να είναι πολύ απλή ή πολύ δύσκολη υπόθεση ανάλογα με το είδος του παιχνιδιού αλλά και τα εργαλεία που θα χρησιμοποιηθούν. Σε αυτό το κεφάλαιο θα αναφέρουμε τα κύρια εργαλεία που θα χρειαστούν για την υλοποίηση ενός παιχνιδιού πάνω στις πλατφόρμες της Microsoft, όπως είναι τα Windows, το Xbox 360 και το windows phone. Οι λόγοι που επιλέχθηκαν οι πλατφόρμες της συγκεκριμένης εταιρίας είναι γιατί τα βασικά εργαλεία παρέχονται δωρεάν και γιατί οι κοινότητες και το υλικό που προσφέρουν υπάρχουν σε αφθονία στο διαδίκτυο.

3.2 Αναφορά στη γλώσσα προγραμματισμού C#

Η C# είναι μια σχετικά νέα αντικειμενοστραφής γλώσσα προγραμματισμού η οποία δημιουργήθηκε από την Microsoft. Δανείζεται πολλά στοιχεία, και έχει παρόμοια σύνταξη, με την C++ και την Java, κάνοντας την εκμάθηση της σχετικά εύκολη. Είναι μία γλώσσα ειδικά σχεδιασμένη για να υποστηρίζει το .NET framework της ίδιας εταιρείας. Βασικό χαρακτηριστικό της είναι ότι δεν παράγει απευθείας κώδικα μηχανής όπως η C++, άλλα ένα ενδιάμεσο κώδικα που στοχεύει το .NET.

Το .NET είναι μια νέα πλατφόρμα ανάπτυξης εφαρμογών σε περιβάλλοντα Windows. Σύμφωνα με την Microsoft, έχει ως σκοπό την απλοποίηση της ανάπτυξης εφαρμογών «κρύβοντας» τις τεχνικές λεπτομέρειες υλοποίησης πολλών λειτουργιών, όπως διαχείριση μνήμης, επικοινωνία μέσω δικτύου, είσοδο/έξοδο από συσκευές και αφήνοντας το προγραμματιστή ελεύθερο να επικεντρωθεί στην «λογική» του προγράμματος. Το .NET χαρακτηρίζεται ως managed πλατφόρμα με την έννοια ότι δημιουργεί ένα ελεγχόμενο και ασφαλές περιβάλλον μέσα στο οποίο μπορεί να τρέξει μια εφαρμογή. Η ασφάλεια έγκειται για παράδειγμα στον έλεγχο στην δέσμευση και προσπέλαση της μνήμης (δεν υπάρχουν pointers, δεν μπορείς να προσπελάσεις μια θέση μνήμης εκτός πίνακα), στο τύπο των μεταβλητών και δεδομένων (δεν μπορείς να θέσεις μια float τιμή σε μια ακέραια μεταβλητή) ή στην αυτόματη υλοποίηση δικλείδων ασφαλείας. Το .NET υποστηρίζει πληθώρα γλωσσών προγραμματισμού οι οποίες είναι ειδικά σχεδιασμένες για αυτό, όπως C#, Visual Basic.NET, J++ και managed C++. Στην πραγματικότητα, το .NET καταλαβαίνει μόνο μια γλώσσα προγραμματισμού την Microsoft Intermediate Language (MSIL). Συνεπώς, οποιαδήποτε γλώσσα προγραμματισμού μπορεί να μεταγλωττιστεί σε MSIL μπορεί να τρέξει στην πλατφόρμα .NET. Ο χρήστης μπορεί ακόμα να γράψει απευθείας ένα

πρόγραμμα σε MSIL στο Notepad να το κάνει compile και να το τρέξει στο .NET. Η καρδιά του .NET αποτελείται λεγόμενο Common Language Runtime (CLR). Η οντότητα αυτή είναι το managed περιβάλλον μέσα στο οποίο τρέχουν οι εφαρμογές .NET. Κατά μια έννοια κρύβει το δύσχρηστο Win32 API που χρησιμοποιείται συχνά για προγραμματισμό εφαρμογών Windows και παρουσιάζει στο χρήστη ένα απλούστερο και περισσότερο εύχρηστο.

Επίσης το .NET παρέχει και μια πληθώρα βιβλιοθηκών με έτοιμες λειτουργίες που μπορεί να χρησιμοποιήσει ο χρήστης για την ανάπτυξη των εφαρμογών. Ο κώδικας που εκτελείται στο CLR έρχεται υπό μορφή assemblies, με επέκταση .dll ή .exe (δεν έχουν σχέση με τα κλασσικά .dll αρχεία των Windows). Το .NET, όπως και η Java χαρακτηρίζονται από την λεγόμενη Just in Time μεταγλώττιση. Ο κώδικας, στη γλώσσα προγραμματισμού που χρησιμοποιεί ο χρήστης, μεταγλωττίζεται αρχικά σε MSIL η οποία αποθηκεύεται σε ένα εκτελέσιμο .exe αρχείο ή σε μια βιβλιοθήκη .dll. Όταν ο χρήστης τρέξει το πρόγραμμα που ανέπτυξε, το CLR διαβάζει το MSIL κώδικα του αρχείο και Just In Time (JIT) το μεταγλωττίζει σε κώδικα Windows (native) έτοιμο προς εκτέλεση, και στην συνέχεια τον εκτελεί. Αυτό το επιπλέον βήμα πριν την εκτέλεση του κώδικα διαφοροποιεί μια managed εφαρμογή σε .NET από μια unmanaged σε C++ για παράδειγμα. Το πρόγραμμα σε C++ είναι ήδη μεταγλωττισμένο σε native Windows κώδικα και τρέχει απευθείας.

Ένα ακόμα χαρακτηριστικό του CLR είναι η αυτοματοποιημένη διαχείριση μνήμης. Σε κλασσικές γλώσσες προγραμματισμού (unmanaged) όπως η C++, όταν ο χρήστης δεσμεύσει μια ποσότητα μνήμης για να αποθηκεύσει ένα αντικείμενο πρέπει να είναι πολύ προσεκτικός στο να την αποδεσμεύσει, να την επιστρέψει στο σύστημα δηλαδή, όταν δεν την χρειάζεται. Αν το αγνοήσει αυτό συστηματικά, τότε θα δημιουργηθεί το επονομαζόμενο memory leak, δηλαδή η διαθέσιμη μνήμη του συστήματος θα ελαττώνεται διαρκώς και σε κάποιο σημείο τα Windows στερέψουν από ελεύθερη μνήμη. Αντιθέτως το CLR προσφέρει ένα μηχανισμό Garbage Collection. Ο χρήστης μπορεί να ζητήσει όση μνήμη χρειάζεται από το σύστημα και να μην ασχοληθεί με την απελευθέρωση της. Ο Garbage Collector υλοποιεί μηχανισμούς που του επιτρέπουν να «καταλάβει» πότε μια δεσμευμένη ποσότητα μνήμης δεν χρησιμοποιείται πλέον και αυτόματα την απελευθερώνει για μετέπειτα χρήση.

Πλεονεκτήματα του .NET

Το .NET έχει πολλά πλεονεκτήματα για την ανάπτυξη εφαρμογών:

- Είναι εγγενώς αντικειμενοστραφής πλατφόρμα.
- Είναι ανεξάρτητο από γλώσσα προγραμματισμού. Σε μια εφαρμογή ένας προγραμματιστής μπορεί να γράφει κώδικα σε C#, άλλος σε VB.NET και άλλος σε managed C++ και τα τμήματα που αναπτύσσει ο καθένας να συνεργάζονται μεταξύ τους χωρίς προβλήματα.
- Η χρήση βιβλιοθηκών (assemblies) κάνει πολύ εύκολη την επαναχρησιμοποίηση κώδικα.
- Παρέχει πολύ εύκολη εγκατάσταση. Αρκεί να αντιγράψουμε το κατάλογο της εφαρμογής σε ένα άλλο υπολογιστή και αυτή θα τρέξει άμεσα. Δεν υπάρχει installation, δεν πειράζει το registry.
- Παρέχει πληθώρα έτοιμων λειτουργιών που κάνουν την ανάπτυξη κώδικα πολύ εύκολη.
- Αυτοματοποιημένη διαχείριση μνήμης, ο χρήστης δεν χρειάζεται να ασχοληθεί με αποδέσμευση μνήμης.

Μειονεκτήματα του .NET (για ανάπτυξη παιχνιδιών)

Το .NET έχει 2 μειονεκτήματα που αφορούν ειδικά την ανάπτυξη βιντεοπαιχνιδιών και όχι την γενική ανάπτυξη εφαρμογών:

- Αυτοματοποιημένη διαχείριση μνήμης, ο χρήστης δεν χρειάζεται να ασχοληθεί με αποδέσμευση μνήμης.
- Το CLR εισάγει μια (μικρή ίσως) καθυστέρηση στην εκτέλεση της εφαρμογής.

Το πρώτο, αν και είναι πολύ χρήσιμο χαρακτηριστικό για γενικό προγραμματισμό, σε ένα βιντεοπαιχνίδι μπορεί να δημιουργήσει πρόβλημα. Η απελευθέρωση μνήμης που δεν χρησιμοποιείται πλέον είναι μια ακριβή εργασία (από πλευράς χρόνου), η οποία επιπλέον είναι μη-ντετερμινιστική, μπορεί να συμβεί δηλαδή οποτεδήποτε. Αυτό μπορεί να έχει σαν αποτέλεσμα ένα παιχνίδι που τρέχει σε ένα σταθερό ρυθμό 60 καρέ το δευτερόλεπτο, να δει μια δραματική πτώση στο ρυθμό ανανέωσης για 1 δευτερόλεπτο, πράγμα ανεπίτρεπτο στην ανάπτυξη βιντεοπαιχνιδιών.

Το δεύτερο αφορά τη Just in Time μεταγλώττιση που υποστηρίζει το CLR. Από τη μία εισάγει μια καθυστέρηση στην εκκίνηση του παιχνιδιού μιας και πρέπει να μεταγλωττιστεί ο κώδικας από την άλλη ο μεταγλωττιστής ο ίδιος δεν είναι βέλτιστος με την έννοια ότι δεν παράγει το καλύτερο δυνατό native κώδικα για Windows.Και τα δυο προβλήματα αυτά μπορούν να αντιμετωπιστούν αν τα λάβουμε υπόψη κατά την ανάπτυξη της εφαρμογής.

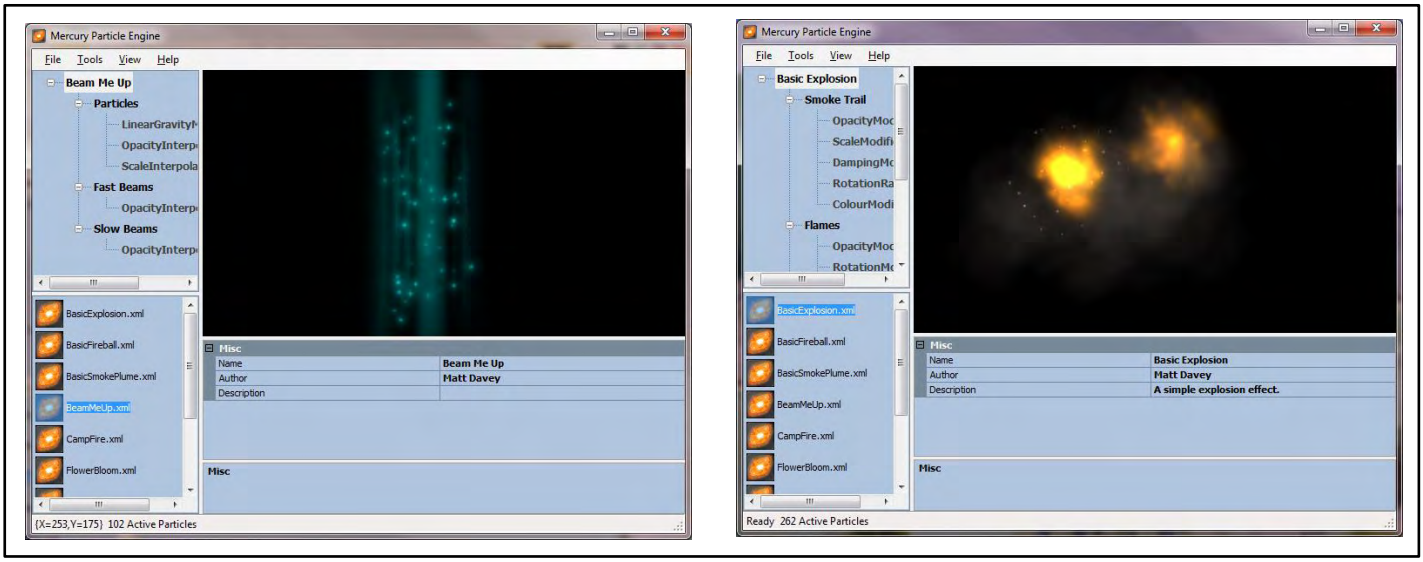
3.3 Αναφορά στο XNA development framework

XNA γενικά ονομάζεται όλο το σετ των εργαλείων που παρέχει η Microsoft για την ανάπτυξη παιχνιδιών σε Windows, Xbox360 και windows phone. Περιλαμβάνει εργαλεία όπως το PIX για γραφικά, το XACT για ήχο, το Xbox Development Kit, το Visual Studio κλπ. Το XNA Game Studio είναι ένα υποσύνολο των εργαλείων αυτών που στοχεύει αποκλειστικά στην ανάπτυξη παιχνιδιών σε Windows, Xbox360 και windows phone χρησιμοποιώντας τη τεχνολογία .NET της ίδιας εταιρίας. Βασίζεται πάνω στο περιβάλλον ανάπτυξης Visual Studio, την γλώσσα προγραμματισμού C#, σε μια ειδική έκδοση του DirectX και φυσικά στο .NET. Το XNA Game Studio έχει φτάσει στην έκδοση 4.0 και είναι διαθέσιμο δωρεάν σε όλους.

Η χρήση του περιβάλλοντος ανάπτυξης του XNA Game Studio έχει πολλά πλεονεκτήματα ειδικά για εκμάθηση προγραμματισμού βιντεοπαιχνιδιών. Καταρχάς έχει διάφορα επίπεδα δυσκολίας. Αν κάποιος επιθυμεί να αναπτύξει ένα διδιάστατο παιχνίδι τότε δεν χρειάζεται να ασχοληθεί καθόλου με μετασχηματισμούς πινάκων, shaders και τριδιάστατα μοντέλα. Αν κάποιος θέλει να δοκιμάσει με κάποιο τριδιάστατο παιχνίδι τότε το XNA Game Studio τον διευκολύνει σημαντικά με αρκετές έτοιμες βιβλιοθήκες που αναλαμβάνουν να φορτώσουν τα μοντέλα από το δίσκο και να τα απεικονίσουν χρησιμοποιώντας έτοιμους shaders. Και αν κάποιος αισθάνεται έτοιμος να πάρει τον έλεγχο του συστήματος μπορεί να παρακάμψει κάθε έτοιμη λειτουργία του XNA Game Studio και να την υλοποιήσει εξαρχής όπως επιθυμεί αυτός. Το ποιο σημαντικό είναι ότι XNA Game Studio μας παρέχει ένα γρήγορο τρόπο να φορτώσουμε και να απεικονίσουμε περιεχόμενο στο παιχνίδι μας, το οποίο είναι αρκετά δύσκολο να το υλοποιήσεις εξαρχής και ένα καλοφτιαγμένο game loop το οποίο μπορούμε να χρησιμοποιήσουμε εύκολα και άμεσα. Επιπλέον το παιχνίδι που αναπτύσσουμε μπορεί να τρέξει με ελάχιστες αλλαγές και σε Windows και σε Xbox360 και σε windows phone.

3.4 Αναφορά στο Mercury Particle Engine 3.1

Το Mercury Particle Engine είναι ένα open source game engine το οποίο αναπτύχθηκε από τον Matthew Davey με σκοπό την δημιουργία δισδιάστατων ειδικών εφέ. Το λογισμικό είναι γραμμένο σε C# και είναι συμβατό με το XNA 4 της Microsoft. Η εφαρμογή είναι πλέον σε stable version και είναι πολύ εύκολη για κάποιον να μάθει να την χρησιμοποιεί και να φτιάξει τα δικά του ειδικά εφέ. Η εφαρμογή διαθέτει παραθυρικό menu και ο χειρισμός της γίνεται με το ποντίκι και το πληκτρολόγιο. Υπάρχουν είδη κάποια έτοιμα εφέ τα οποία μπορεί να χρησιμοποιήσει ο χρήστης και να τα παραμετροποιήσει κατά το δοκούν ή μπορεί να φτιάξει από την αρχή ένα δικό του εφέ. Στην εικόνα 7 παρακάτω βλέπουμε μερικά στιγμιότυπα από την χρήση του mercury particle engine.



εικόνα 7

Το επόμενο πράγμα που θα κάνουμε είναι να αναλύσουμε το πώς λειτουργεί ένα particle engine και ποια τα κύρια χαρακτηριστικά του. Ένα particle effect συνήθως αποτελείται από τρία βασικά μέρη: το σωματίδιο (particle), τον εκτοξευτή σωματιδίων (particles emitter) και τη μηχανή (engine).

- **Το σωματίδιο (Particle)**
  Είναι ένα μικρό σημείο στην οθόνη το οποίο κάποια στιγμή στο μέλλον θα ζωγραφιστεί με μία εικόνα. Τις περισσότερες φορές έχει συγκεκριμένη θέση μέσα στο παιχνίδι, καθώς επίσης και ταχύτητα (velocity), γωνία (angle) σύμφωνα με την οποία περιστρέφεται, γωνιακή ταχύτητα (angular velocity) η οποία καθορίζει πόσο γρήγορα περιστρέφεται το σωματίδιο, εικόνα (texture) και χρώμα (color). Το εφέ μας αποτελείται από πολλά τέτοια σωματίδια τα οποία όλα μαζί συνθέτουν ένα πολύ ενδιαφέρον οπτικό αποτέλεσμα. Επίσης τα σωματίδια συνήθως έχουν και έναν προκαθορισμένο χρόνο ζωής (life time) σύμφωνα με τον οποίο γεννιούνται και αφαιρούνται για να δώσουν την θέση τους σε άλλα καινούργια
- **Ο εκτοξευτής σωματιδίων (Particles Emitter)**
  Καθορίζει επί της ουσίας την θέση και τον τρόπο με τον οποίο θα παράγονται τα σωματίδια καθώς επίσης και το μέγιστο και ελάχιστο πλήθος των σωματιδίων που μπορεί να υπάρχουν κάθε στιγμή.
- **Η μηχανή (Engine)**
  Η μηχανή στην ουσία αποτελεί το δομικό στοιχείο του εφέ καθώς είναι υπεύθυνο για την κατάσταση των προηγούμενων δύο συστατικών μερών.

Το Mercury Particle engine λοιπόν μας δίνει την δυνατότητα να δημιουργήσουμε, μέσα από εικόνες (textures) και αλλάζοντας τις ιδιότητες των παραπάνω κύριων στοιχείων ενός special effect, το δικό μας ειδικό εφέ όπως ο έχουμε φανταστεί. Στη συνέχεια μπορούμε να το αποθηκεύσουμε σε μορφή “XML” και να το φορτώσουμε μέσα στο παιχνίδι μας, όπου και μπορούμε να το συνδέσουμε με κάποιο γεγονός και να το ενεργοποιήσουμε.

4. Βασικές τεχνικές υλοποίησης ενός 2D Video Game

4.1 Εισαγωγή στην κύρια δομή ενός XNA project

Αφού έχουμε τελειώσει με τον σχεδιασμό του παιχνιδιού και έχοντας κάνει μία μικρή αναφορά στην γλώσσα προγραμματισμού C# και στο εργαλείο XNA μπορούμε πλέον να προχωρήσουμε στο στάδιο της υλοποίηση και του προγραμματισμού. Αν ρίξουμε μία ματιά στην δομή ενός XNA project θα δούμε ότι αποτελείται από πέντε βασικά μέρη: το initialisation, το content loading, το content unloading, το updating και το drawing. Καθένα από αυτά υλοποιείται σαν μία συνάρτηση ενός αντικειμένου τύπου game (game object class) και αυτές οι μέθοδοι τις περισσότερες φορές καλούνται αυτόματα από το XNA. Επίσης είναι καλό ως μοντέλο προγραμματισμού να κρατάμε αυτή την δομή και για κάθε καινούργια κλάση που υλοποιούμε.

Μέθοδος Initialize

Καλείται στην αρχή της εκτέλεσης του προγράμματος και χρησιμοποιείται για να αρχικοποιήσει τις εκάστοτε μεταβλητές με τις τιμές που έχουν οριστεί από τον προγραμματιστή. Η μέθοδος initialize καλεί και την LoadContent γι αυτό και πολλές φορές η initialize δεν χρησιμοποιείται άμεσα, αφού πολλοί προγραμματιστές προτιμούν να αρχικοποιούν τις μεταβλητές μαζί με την φόρτωση του περιεχομένου μέσα από την LoadContent.

Μέθοδος LoadContent

Αυτή η μέθοδος χρησιμοποιείται προκειμένου να φορτώσουμε στην μνήμη το περιεχόμενο του παιχνιδιού (γραφικά, ήχους κτλ.) και να είναι έτοιμο προς χρήση όταν αυτό ζητηθεί. Σε γενικές γραμμές, μπορούμε να φορτώσουμε όλο το περιεχόμενο στη μέθοδο LoadContent που έχουμε δημιουργήσει και να ξεμπερδεύουμε έτσι απλά, αλλά σε μεγαλύτερα projects που οι απαιτήσεις σε μνήμη είναι ακόμη μεγαλύτερες, καλό είναι να υλοποιήσουμε κάτι πιο κομψό όπου η φόρτωση του κάθε περιεχομένου θα γίνεται μόνο όταν αυτό κρίνεται απαραίτητο μέσα από κάποιο load screen. Το XNA καλεί αυτόματα τη μέθοδο LoadContent την κατάλληλη στιγμή κατά την έναρξη του παιχνιδιού για να φορτώσει το αρχικό ή όλο το περιεχόμενο.

Μέθοδος UnloadContent

Η εκφόρτωση του περιεχομένου κατά κύριο λόγο μπορεί να αγνοηθεί, καθώς ο content manager του XNA αναλαμβάνει να απομακρύνει από την μνήμη αυτόματα το περιεχόμενο κατά τον τερματισμό της εφαρμογής. Σε πιο ειδικές περιπτώσεις όπου θέλουμε να κρατήσουμε συγκεκριμένα δεδομένα στη μνήμη και να μην αντικατασταθούν όταν αυτή γεμίσει, καλό είναι να χρησιμοποιούμε την UnloadContent όταν το περιεχόμενο δεν πρόκειται να χρησιμοποιηθεί σύντομα.

Μέθοδος Update

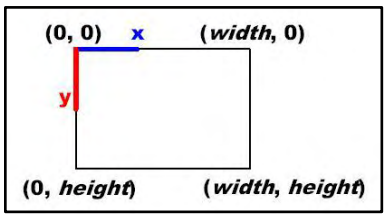
Η μέθοδος Update είναι εκεί που η λογική του παιχνιδιού παίρνει σάρκα και οστά. Από εδώ ενεργοποιούνται οι ήχοι, λαμβάνεται η είσοδος του χειρισμού και αλληλεπιδρούν τα αντικείμενα του παιχνιδιού. Παίρνει ως παράμετρο την μεταβλητή GameTime, η οποία είναι καλό να χρησιμοποιείται από τον κωδικό του εκάστοτε προγραμματιστή καθώς έτσι το λειτουργικό κομμάτι του παιχνιδιού θα είναι ανεξάρτητο από το frame rate. Αυτό θα εξασφαλίσει μια ομαλή λειτουργία στην απεικόνιση του παιχνιδιού, αφού οι κινήσεις των αντικειμένων θα έχουν προβλέψιμη συμπεριφορά σε όλες τις ταχύτητες καρέ. Το XNA καλεί αυτόματα αυτή τη μέθοδο σε κάθε κύκλο εκτέλεσης του προγράμματος.

Μέθοδος Draw

Η μέθοδος Draw είναι υπεύθυνη για την απεικόνιση των αντικειμένων του παιχνιδιού στην οθόνη. Αυτή η μέθοδος υπό φυσιολογικές συνθήκες καλείται 30 φορές ανά δευτερόλεπτο, όσο δηλαδή και το frame rate που υποστηρίζει το XNA. Το XNA προσπαθεί να κρατήσει σταθερό το frame rate αλλά όταν αυτό δεν είναι εφικτό, η μέθοδος αυτή προσπερνιέται (frame skipping).

4.2 Γραφικά 2D

Το πρώτο στάδιο υλοποίησης ενός video game είναι να εμφανίσουμε στην οθόνη μία δισδιάστατη εικόνα. Αρχικά θα πρέπει να γίνει αντιληπτό από τον προγραμματιστή το δισδιάστατο σύστημα συντεταμένων που χρησιμοποιεί το XNA.



εικόνα 8

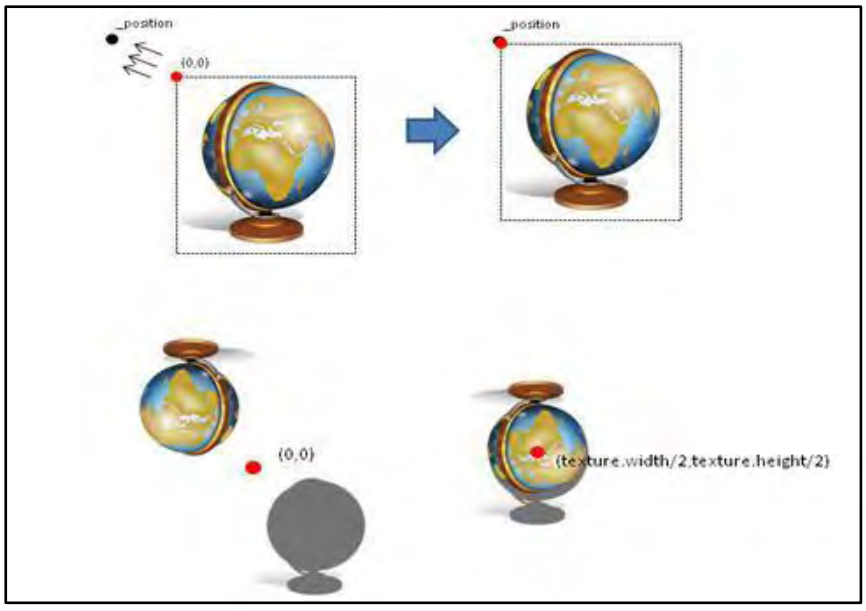
Στην εικόνα 8 παρατηρούμε ότι το σημείο (0, 0) βρίσκεται πάνω αριστερά, ενώ το width και το height είναι η ανάλυση που έχουμε ορίσει ότι θα τρέχει το παιχνίδι.

Μέσα σε αυτόν τον δισδιάστατο κόσμο είναι που θα λάβει χώρα και το παιχνίδι μας. Οι δισδιάστατες εικόνες είναι γνωστές και ως Sprites. Τα spites μπορεί να ανήκουν στον δισδιάστατο χώρο αλλά έχουν εφαρμογές και στον τρισδιάστατο. Παρακάτω ακολουθούν τα βασικά βήματα που χρειάζονται για να εμφανιστεί ένα sprite στην οθόνη.

1. Αρχικοποιούμε την συσκευή μας και φορτώνουμε το sprite στην μνήμη.
   SpriteBatch spriteBatch = new SpriteBatch(GraphicsDevice);
   private Texture2D aSprite = Content.Load<Texture2D>("sprite.jpg");

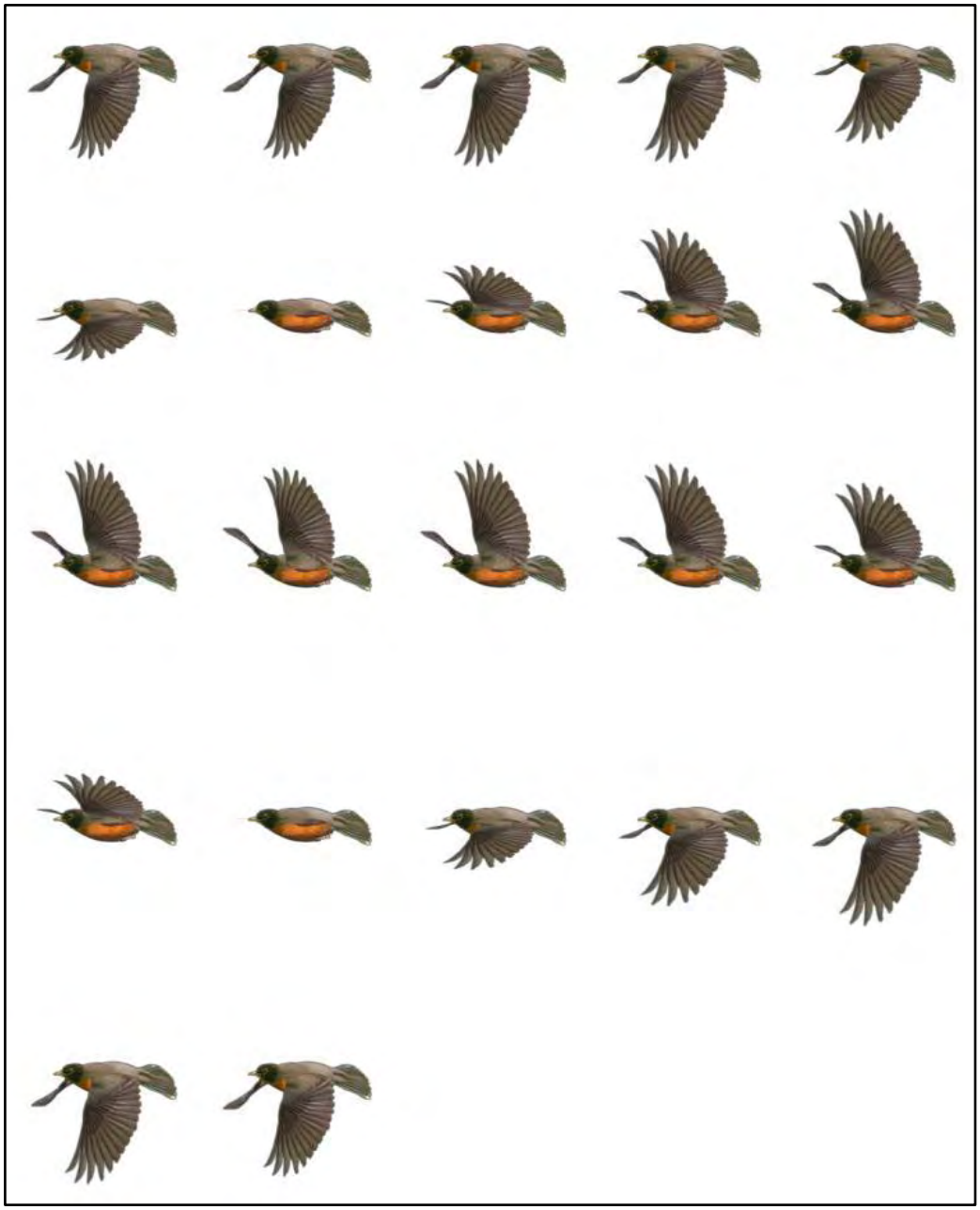
2. Εμφανίζουμε το sprite μας στην οθόνη μέσω της μεθόδου Draw.
   spriteBatch.Begin();
   spriteBatch.Draw(aSprite, new Rectangle(0, 0, 128, 128), Color.White);
   spriteBatch.End();

Αυτό που αξίζει να προσέξουμε από τον παραπάνω κώδικα είναι το πώς συντάσσεται η μέθοδος Draw, να σημειωθεί ότι αυτή είναι η πιο απλή της μορφή. Οι παράμετροι που δέχεται είναι το sprite τύπου Texture2D που θέλουμε να απεικονίσουμε, ένα αντικείμενο τύπου Rectangle όπου μέσα σε αυτό θα εμφανιστεί η εικόνα μας και η απόχρωση που θέλουμε να του δώσουμε. Το αντικείμενο τύπου Rectangle είναι υπεύθυνο τόσο για την θέση όσο και για το μέγεθος της εικόνας. Για παράδειγμα στον παραπάνω κώδικα η εικόνα μας θα εμφανιστεί στο σημείο (0,0) (πάνω αριστερό άκρο της εικόνας) και θα έχει διαστάσεις 128x128. Αμέσως λοιπόν γίνεται αντιληπτό ότι εάν θέλουμε να δώσουμε κίνηση ή να αυξομειώσουμε την εικόνα μας ότι θα πρέπει μέσω της μεθόδου update να αλλάζουμε της ιδιότητες του ορθογωνίου. Τώρα εάν θέλουμε να περιστρέφουμε την εικόνα μας, το μόνο που πρέπει να προσθέσουμε στην Draw είναι το σημείο και τις μοίρες περιστροφής. Ένα παράδειγμα μπορούμε να δούμε παρακάτω στην εικόνα 9.



εικόνα 9

Επίσης η μέθοδος Draw μας δίνει την δυνατότητα προσθέτοντας άλλο ένα αντικείμενο τύπου Rectangle ως παράμετρο, να εμφανίσουμε στην οθόνη ένα κομμάτι του sprite. Αυτή η λειτουργία είναι πολύ σημαντική καθώς με αυτόν τον τρόπο μπορούμε να δημιουργήσουμε animation.



εικόνα 10

Στην εικόνα 10 βλέπουμε κάτι το οποίο είναι γνωστό και ως sprite sheet και στην ουσία εμπεριέχει όλα τα καρέ μια ολοκληρωμένης κίνησης. Ο προγραμματιστής επιλέγοντας και εναλλάσσοντας την εικόνα με το επόμενο καρέ πολύ γρήγορα επιτυγχάνει την ομαλή προβολή της κίνησης, στο συγκεκριμένο παράδειγμα το φτερούγισμα ενός κοκκινολαίμη.

4.3 Χειρισμός

Ο χειρισμός ενός επιτυχημένου παιχνιδιού θα πρέπει να είναι ακριβής και προβλέψιμος. Το χειριστήριο που μας παρέχει η Microsoft (xbox 360 controller) είναι σίγουρα ένα πολύ καλό εργαλείο, αλλά αυτό από μόνο του δεν κάνει τίποτα. Ο τρόπος που θα προγραμματίσουμε το παιχνίδι για να ακούει της εντολές που θα του δίνει ο χρήστης είναι ίσως το πιο σημαντικό κομμάτι. Με το XNA η Microsoft επί της ουσίας δημιούργησε μια πλατφόρμα που καθιστά πολύ εύκολο σε κάποιον να δημιουργήσει ένα παιχνίδι για PC και για Xbox 360 ταυτόχρονα. Αυτό συμβαίνει γιατί το XNA παρέχει πολύ πιο εύκολη πρόσβαση στο hardware από ότι ο προκάτοχός του DirectX. Αυτό συμβαίνει και με το Xbox 360 controller, αφού μέσω του XNA η δουλειά του προγραμματιστή γίνεται πολύ πιο εύκολη.

Στην εικόνα 11 βλέπουμε το χειριστήριο του Xbox 360.

εικόνα 11

Οι είσοδοι που δέχεται το χειριστήριο είναι είτε ψηφιακές είτε αναλογικές. Οι ψηφιακές είσοδοι έχουν δύο καταστάσεις για να σηματοδοτήσουν εάν έχουν ενεργοποιηθεί ή όχι. *ButtonState.Pressed ή ButtonState.Released.* Οι ψηφιακές είσοδοι στο χειριστήριο είναι τα κουμπιά A, B, X, Y, LB , RB, Back, Start, τα κουμπιά κάτω από τους μοχλούς (Thumbsticks) και ο σταυρός (D-Pad). Η κατάσταση στις αναλογικές εισόδους αναπαρίσταται από μεταβλητές κινητής υποδιαστολής (float). Οι σκανδάλες (triggers) στο χειριστήριο παράγουν τιμές από 0.0 (κανονική κατάσταση) έως 1.0 (πατημένες πλήρως). Οι δύο μοχλοί (thumbsticks) από την άλλη παράγουν τιμές από -1.0 έως +1.0 για τους άξονες X και Y, όπου 0 το κέντρο των αξόνων. Το κεντρικό κουμπί με το "X" είναι προσβάσιμο μόνο από το λειτουργικό σύστημα και δεν είναι διαθέσιμο στον προγραμματιστή. Ο χειρισμός των εισόδων γίνεται κάθε φορά στην συνάρτηση Update.

Η κατάσταση του χειριστηρίου είναι προσβάσιμη από την κλάση Microsoft.XNA.Framework.Input.GamePad. Αυτή η κλάση διαθέτει μόλις τρείς

μεθόδους, την GetCapabilities, την GetState, και την SetVibration και θεωρώ ότι από τα ονόματα τους και μόνο ότι είναι αυτονόητη η λειτουργία τους. Και οι τρείς μέθοδοι δέχονται ως παράμετρο μία μεταβλητή τύπου PlayerIndex, σύμφωνα με την οποία καθίσταται σαφές για ποιανού παίχτη το χειριστήριο κλήθηκε η εκάστοτε συνάρτησης. Ο παρακάτω κώδικας αλλάζει την τιμή της Boolean μεταβλητής σε true αν το κουμπί Α είναι πατημένο ή σε false στην άλλη περίπτωση.

```
this.gamePadState = GamePad.GetState(this.playerIndex);

boolean buttonAState  = (this.gamePadState.Buttons.A == Input.ButtonState.Pressed);
```

Το χειριστήριο του Xbox 360 είναι εξοπλισμένο με δύο μικρούς κινητήρες για να επιτυγχάνει δόνηση. Ο αριστερός κινητήρας προκαλεί αργή και έντονη δόνηση, ενώ ο δεξής πιο μικρή και απότομη. Όπως αναφέραμε και προηγουμένως η δόνηση ενεργοποιείται μέσω της μεθόδου *SetVibration* έχοντας ως ορίσματα, εκτός από το PlayerIndex, και δύο float μεταβλητές που ορίζουν την ένταση του κάθε κινητήρα. Ο κώδικας που ακολουθεί ενεργοποιεί τον αριστερό κινητήρα σε μέτρια δόνηση.

```
GamePad.SetVibration(playerIndex, 0.5f, 0.0f);
```

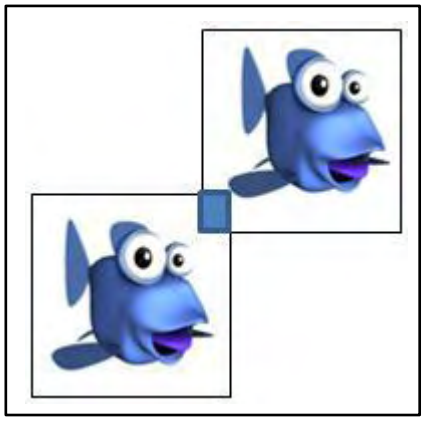
4.4 Ανίχνευση συγκρούσεων

Σε αυτό το κεφάλαιο θα αναφερθούμε στις βασικές τεχνικές ανίχνευσης συγκρούσεων αντικειμένων ή αλλιώς Sprite. Η πρώτη και η πιο βασική τεχνική είναι αυτή του περιβάλλοντος σχήματος. Η λογική είναι απλή, για κάθε αντικείμενο βρίσκουμε ένα βασικό σχήμα που να το εμπεριέχει. Η ανίχνευση συγκρούσεων βασίζεται σε αλγορίθμους που εντοπίζουν πότε τέμνονται δύο βασικά γεωμετρικά σχήματα και μπορούν να βρεθούν πολύ εύκολα στη βιβλιογραφία. Τα βασικά σχήματα συνήθως είναι τετράγωνα, ορθογώνια, τρίγωνα ή κύκλοι. Η εικόνα 12 δείχνει πως μπορεί να επιτευχθεί αυτό.

εικόνα 12

Η μέθοδος αυτή είναι πάρα πολύ γρήγορη και δεν χρειάζεται πολύπλοκους υπολογισμούς, παρόλα αυτά όπως είναι φανερό υστερεί σε ακρίβεια. Για να πετύχουμε περισσότερη ακρίβεια μπορούμε να κάνουμε δύο πράγματα:

- Να χρησιμοποιήσουμε παραπάνω από ένα βασικό σχήμα για να περιβάλλουμε το αντικείμενο μας.
- Να χρησιμοποιήσουμε ένα βασικό σχήμα που να περιβάλει το αντικείμενο μας και όταν διαπιστωθεί σύγκρουση να χρησιμοποιήσουμε per-pixel ανίχνευση.



εικόνα 13

Στο παράδειγμα της εικόνας 13, αρχικά ο αλγόριθμος ελέγχει κατά πόσο υπάρχει σύγκρουση μεταξύ των δύο τετραγώνων. Εάν υπάρχει (όπως στην περίπτωσή μας) τότε θα κληθεί να τρέξει ο per-pixel αλγόριθμός μόνο για το επικαλυπτόμενο κομμάτι (μπλε ορθογώνιο στο παράδειγμα μας). Εάν κάποιο pixel από την μία εικόνα επικαλύπτει κάποιο pixel της άλλης εικόνας, τότε ο αλγόριθμος επιστρέφει true ανιχνεύοντας έτσι την σύγκρουση. Δεν πρέπει να ξεχνάμε βέβαια ότι ακόμα και με αυτή την βελτίωση ο per-pixel αλγόριθμος είναι πολύ αργός και θα πρέπει να χρησιμοποιείται αυστηρά μόνο σε περιπτώσεις που χρειάζεται απόλυτη ακρίβεια και είναι πολύ δύσκολο να επικαλύψουμε το αντικείμενό μας με μικρότερα βασικά σχήματα.

4.5 Μουσική και ηχητικά εφέ

Όπως σε μία κινηματογραφική ταινία, έτσι και σε ένα παιχνίδι η μουσική ευθύνεται σε ένα πολύ μεγάλο κομμάτι για την επιτυχία και το αντίκτυπο που θα προκαλέσει στο κοινό το οποίο απευθύνεται. Δεν ξέρω αν θα γνωρίζαμε σήμερα τις ταινίες του Quentin Tarantino ή βιντεοπαιχνίδια όπως το Medal Gear Solid αν δεν είχαν τόσο δυνατή μουσική και ηχητικά εφέ. Στο κεφάλαιο αυτό δεν θα ασχοληθούμε με το πώς φτιάχνει κάποιος ένα καλό soundtrack, αυτό ξεφεύγει κατά πολύ από την επιστήμη των υπολογιστών αλλά δεν ξέρω και κατά πόσο έχει βρεθεί η μυστική φόρμουλα για την δημιουργία ενός επιτυχημένου μουσικού κομματιού. Εκεί που θα επικεντρωθούμε είναι στο πως φορτώνουμε και πως αναπαράγουμε τη μουσική και τα ηχητικά εφέ που θέλουμε να χρησιμοποιήσουμε στο παιχνίδι μας.

Για να προσθέσουμε λοιπόν αρχεία ήχου στο project μας ακολουθούμε ακριβώς την ίδια διαδικασία που θα εκτελούσαμε αν θέλαμε να προσθέσουμε οποιουδήποτε άλλου είδους Content (Εικόνες, γραμματοσειρές κτλ.). Η φόρτωση στη μνήμη γίνεται λοιπόν με την παρακάτω εντολή:

```
SoundEffect effect = Content.Load<SoundEffect>("testSound");
```

Τώρα ένα θέλουμε να αναπαράγουμε αυτό το αρχείο ήχου που μόλις φορτώσαμε αρκεί να καλέσουμε στο κατάλληλο σημείο μέσα από την μέθοδο Update την εντολή effect.Play(). Αντίστοιχα εάν θέλουμε να κάνουμε παύση στον ήχο ή να τον σταματήσουμε τελείως αρκεί να καλέσουμε τις εντολές effect.Pause() και effect.Stop() αντίστοιχα. Τα αντικείμενα τύπου SoundEffect έχουν πάρα πολλές ιδιότητες που μπορούμε να χρησιμοποιήσουμε για να αλλάξουμε την ένσταση του ήχου, το pitch, το pan κτλ. Για να το επιτύχουμε αυτό αρκεί να αλλάξουμε τις τιμές των αντίστοιχων μεταβλητών όπως παρακάτω:

```
effect.Volume = 0.5f;

effect.Pan = -0.5f;

effect.Pitch = 0.5f;
```

Από τον τρόπο με τον οποίο χειριζόμαστε τον ήχο μπορούμε να καταλάβουμε αυτό που λέγαμε σε προηγούμενα κεφάλαια, ότι το XNA μας διευκολύνει σε πολύ μεγάλο βαθμό στο να υλοποιήσουμε ένα video game χωρίς να μας περιπλέκει με πολύπλοκες εντολές και ρυθμίσεις, δίνοντας μας πάντα την δυνατότητα να παραμετροποιήσουμε το παιχνίδι μας και να το φτιάξουμε έτσι όπως το έχουμε φανταστεί. Αυτή είναι και η δύναμη του XNA και ο λόγος για τον οποίο έχει γίνει μία από τις πιο δημοφιλής πλατφόρμες ανάπτυξης βιντεοπαιχνιδιών.

4.6 Ειδικά εφέ

Σε αυτό το κεφάλαιο θα δούμε πώς εμφανίζουμε και ενεργοποιούμε τα ειδικά εφέ που έχουμε δημιουργήσει με το εργαλείο που έχουμε αναφέρει πιο πάνω, δηλαδή το Mercury Particle Engine. Άρα θεωρούμε ότι έχουμε είδη σχεδιάσει το εφέ μας και το έχουμε σε μορφή "XML".

Το πρώτο πράγμα που πρέπει να κάνουμε λοιπόν είναι να φορτώσουμε το XML αρχείο και τα αντίστοιχα textures του ειδικού εφέ μέσα στο "Content" του XNA project μας, δηλαδή μέσα στο παιχνίδι που αναπτύσσουμε καθώς επίσης και να προσθέσουμε το αρχείο "MercuryProject.dll" μέσα στο "Reference" του project. Στη συνέχεια προσθέτουμε τις ακόλουθες βιβλιοθήκες χρησιμοποιώντας το "using statement" όπως παρακάτω:

```
using ProjectMercury;
using ProjectMercury.Emitters;
using ProjectMercury.Modifiers;
using ProjectMercury.Renderers;
```

Στην κλάση προσθέτουμε της μεταβλητές:

```
// Renderer that draws particles to screen
Renderer myRenderer;
// Particle effect object to store the info about particle
ParticleEffect myEffect;
```

Για να αρχικοποιήσουμε το εφέ προσθέτουμε στον constructor:

```
// Create new renderer and set its graphics devide to "this" device
myRenderer = new SpriteBatchRenderer { GraphicsDeviceService = graphics };
myEffect = new ParticleEffect();
```

Στη συνέχεια μέσα στη μέθοδο LoadContent γράφουμε:

```
myEffect = Content.Load("BasicExplosion");
myEffect.LoadContent(Content);
myEffect.Initialise();
myRenderer.LoadContent(Content);
```

Για να το ενεργοποιήσουμε τοποθετούμε στην μέθοδο Update:

```
// get the latest mouse state
MouseState ms = Mouse.GetState();

// Check if mouse left button was presed
 if (ms.LeftButton == ButtonState.Pressed){

        // Add new particle effect to mouse coordinates
        myEffect.Trigger(new Vector2(ms.X, ms.Y));

}

// "Deltatime" ie, time since last update call

float SecondsPassed = (float)gameTime.ElapsedGameTime.TotalSeconds;
myEffect.Update(SecondsPassed);
```

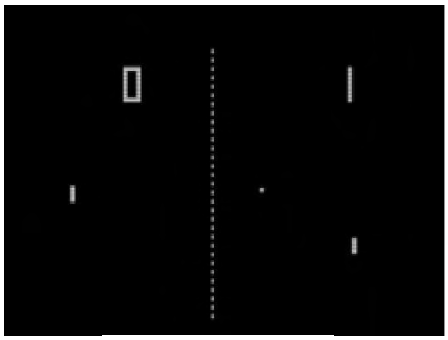
Τέλος για να εμφανίσουμε το εφέ στη μέθοδο Draw προσθέτουμε:

```
myRenderer.RenderEffect(myEffect);
```

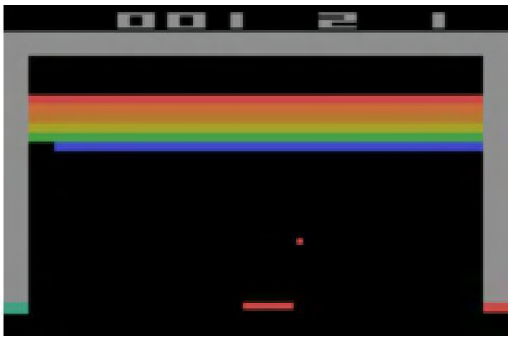
5. Παρουσίαση του παιχνιδιού Bong

5.1 Η αρχική ιδέα και η τελική μορφή του

Το Bong είναι ένα παιχνίδι το οποίο σχεδιάστηκε και υλοποιήθηκε για το Xbox 360 της Microsoft. Η ιδέα για την δημιουργία του Bong βασίστηκε πάνω στα πολύ γνωστά Arcade παιχνίδια Pong και Brick, τα οποία μπορούμε να δούμε παρακάτω στις εικόνες 14 και 15.

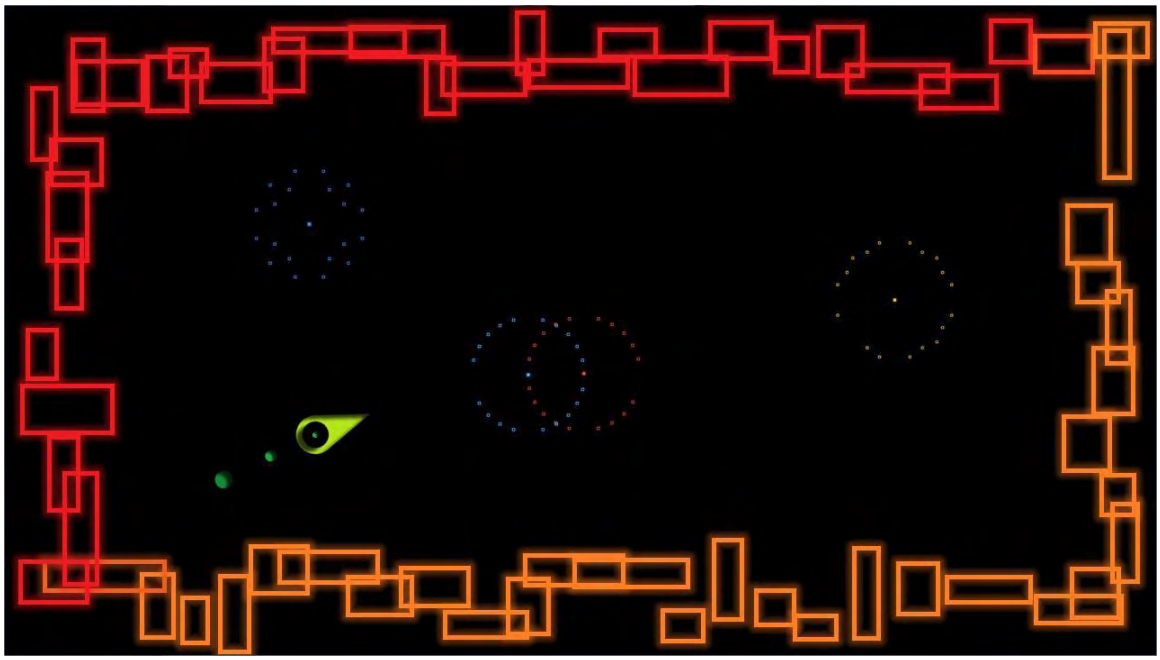


εικόνα 14



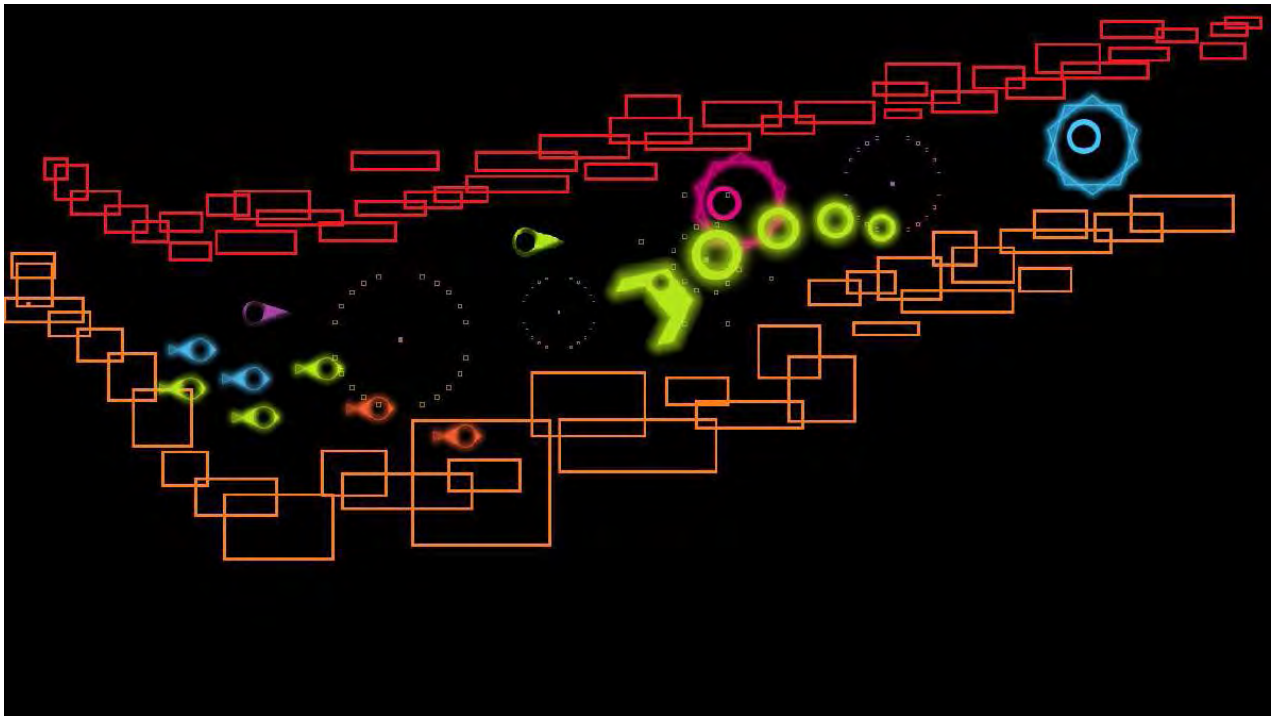
εικόνα 15

Αρχικά η ιδέα του παιχνιδιού Bong ήταν η εξής: ο παίχτης είχε ως στόχο να μαζέψει όλες της "φυσαλίδες" που βρίσκονται στο κέντρο της οθόνης πριν του τελειώσουν τα κινούμενα κουτιά που τον περιβάλουν. Για να γίνει καλύτερα αντιληπτό στην εικόνα 16 βλέπουμε ένα στιγμιότυπο του παιχνιδιού.



εικόνα 16

Στην πορεία η αρχική ιδέα εμπλουτίστηκε από άλλες καλύτερες με απώτερο στόχο το παιχνίδι να γίνει πιο περιπετειώδες και διασκεδαστικό για τον χρήστη. Έτσι πλέον ο παίχτης βρίσκεται σε ένα κινούμενο χώρο και καλείται να φτάσει μέχρι το τέλος της πίστας, χρησιμοποιώντας τα κινούμενα κουτιά που τον περιβάλλουν προκειμένου να παραμείνει εντός των ορίων καθώς και να επιβιώσει από τους εχθρούς που θα συναντήσει όπως δείχνει το στιγμιότυπο της εικόνας 17.



εικόνα 17

Το Bong είναι ένα παιχνίδι το οποίο παίζεται από έναν μέχρι και τέσσερις παίχτες. Κάθε παίχτης αναλαμβάνει τον έλεγχο ενός “Tear”. Η μύτη του Tear χρησιμοποιείται από τον παίχτη για να επιλέξει την πορεία που θέλει να διαγράψει, ενώ το πίσω μέρος του έχει την ιδιότητα να δίνει ταχύτητα στο Tear αναπηδώντας πάνω στα κινούμενα κουτιά που το περιβάλουν. Το Tear είναι εξοπλισμένα με “Boost”, μια ιδιότητα η οποία επιτρέπει στον παίχτη να διόρθωση την πορεία του. Τα πολύχρωμα κουτιά τα οποία έρπουν κατά μήκος της πίστας, βοηθούν τον παίχτη να παραμείνει εντός της οθόνης και εξαφανίζονται μόλις έρθουν σε επαφή με το Tear δίνοντας του μία μικρή ποσότητα “Boost”. Τέλος οι φυσαλίδες που υπάρχουν στο κέντρο της πίστας αποτελούν και τον στόχο των παιχτών καθώς πρέπει να συλλέξουν όσες περισσότερες μπορούν προκειμένου να σημειώσουν ένα υψηλό score.

Όπως αναφέραμε και παραπάνω ο παίχτης καλείται να αντιμετωπίσει και κάποια εμπόδια όπως είναι τα echofishes και τα optagons. Το echofish δεν αποτελεί σημαντική απειλεί για τους παίχτες, αλλά έχει ως στόχο να τους δυσκολέψει στην συλλογή πόντων εάν δεν το εξολοθρεύσουν έγκαιρα, αφού τρέφεται με φυσαλίδες. Ένα optagon από την άλλη μπορεί να σκοτώσει ένα παίχτη ακαριαία. Ένα Tear λοιπόν μπορεί να πεθάνει εάν έρθει σε επαφή με ένα optagon, εκτός και αν προλάβει να το σκοτώσει πρώτο χτυπώντας το με την μύτη του.

Στην εικόνα 18 που ακολουθεί βλέπουμε και της οθόνες του παιχνιδιού καθώς και την σειρά εμφάνισής τους.

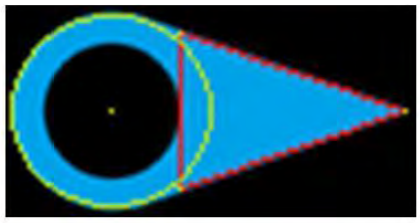
εικόνα 18

5.2 Τα κύρια στοιχεία του παιχνιδιού

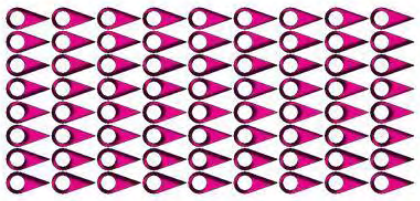
Σε αυτό το σημείο παρουσιάζονται τα βασικά στοιχεία του παιχνιδιού καθώς και μία σύντομη περιγραφή για τις ιδιότητες που τα διέπουν και τον τρόπο με τον οποίο ενορχηστρώθηκαν μέσα στο project.

Το Tear (εικόνα 19)

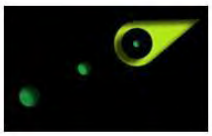
- Τα βασικά σχήματα ανίχνευσης συγκρούσεων του Tear



- Το animation της σκίασης του Tear λόγω της περιστροφής.



- Το "Boost". (Μικραίνει όσο χρησιμοποιείται)



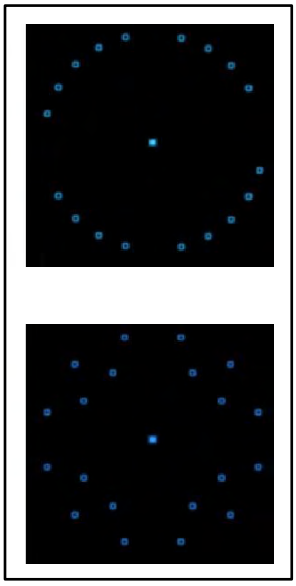
εικόνα 19

Η φυσαλίδα (εικόνα 20)

Η φυσαλίδα αποτελείται από μικρές κουκίδες οι οποίες βρίσκονται τοποθετημένες περιμετρικά μίας κύριας και πιο έντονης κουκίδας, τον πυρήνα.

Ο παίχτης ανταμείβεται με τις κουκίδες με τις οποίες έρχεται σε επαφή. Εάν έρθει σε επαφή με τον πυρήνα τότε ανταμείβεται με όλες τις κουκίδες για την ευστοχία του.

Κάθε κουκίδα δύναται να αλλάξει την απόσταση της από τον πυρήνα ακλουθώντας την μουσική του παιχνιδιού, δημιουργώντας έτσι τυχαίους σχηματισμούς.



εικόνα 20

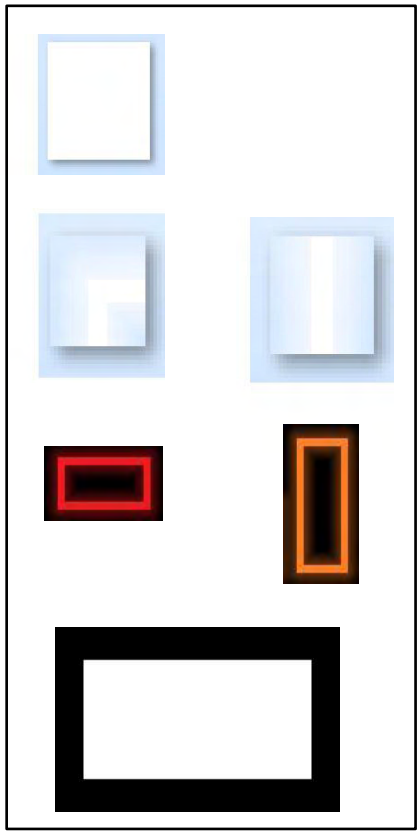
Το κουτί (εικόνα 21)

Το κουτί κατασκευάζεται από την παραμόρφωση ενός λευκού τετραγώνου.

Εικόνα για το περίγραμμα του κουτιού. (Γωνίας και πλευράς)

Το κουτί αυξομειώνει το μέγεθός του κατά τους άξονες x και y, δίνοντας την αίσθηση ότι έρπεται.

Η κύρια οθόνη του παιχνιδιού. Η μαύρη περιοχή είναι το πλαίσιο μέσα στο οποίο κινούνται τα κουτιά στις πίστες bonus ή ακολουθούν την διαδρομή που έχει οριστεί από τον level editor.



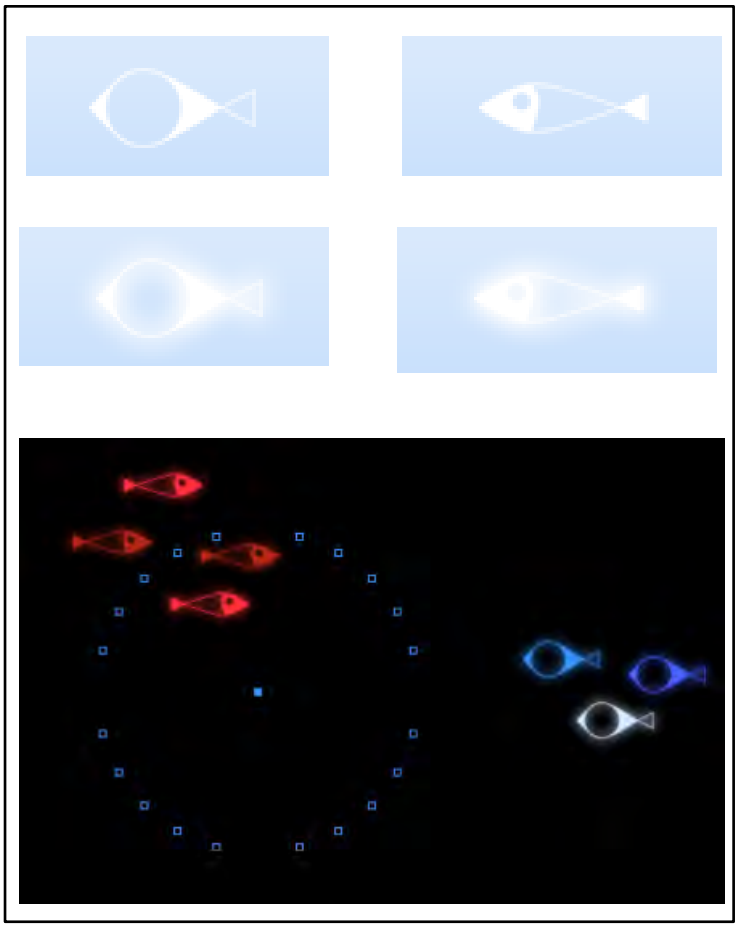
εικόνα 21

Το Echofish (εικόνα 22)

Τα echofishes είναι δύο τύπων και είναι αυτά τα οποία βλέπουμε δεξιά.

Τόσο αυτά όσο και τα περιγράμματά τους έχουν λευκό χρώμα και για να μπορούν να αλλάζουν από τον level editor

Τα echofishes κινούνται σε ομάδες στον οριζόντιο άξονα. Η ταχύτητα τους και η απόσταση που θα διανύσουν ορίζεται από τον level editor. Τρέφονται με φυσαλίδες και πεθαίνουν εάν έρθουν σε επαφή με κάποιο Tear.



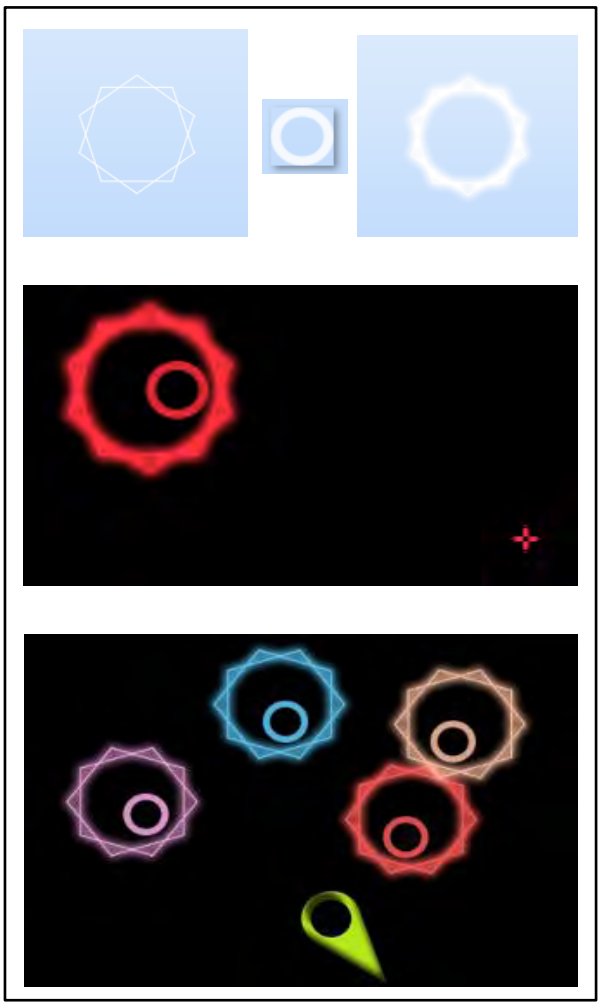
εικόνα 22

Το Optagon (εικόνα 23)

Τα optagons όπως και τα περισσότερα στοιχεία του παιχνιδιού είναι λευκά για να μπορούν αργότερα να πάρουν οποιοδήποτε χρώμα και αποτελούνται από το κύριο σχήμα τους, το μάτι και το περίγραμμά τους.

Κάθε optagon έχει και ένα target, ένα σημείο δηλαδή το οποίο μαζί με την αρχική του θέση ορίζει την ευθύγραμμη πορεία την οποία θα ακολουθήσει.

Τα optagons κοιτάζουν πάντα το ποιο κοντινό τους Tear και γίνονται όλο και πιο φωτεινά όσο το πλησιάζουν. Επίσης αποτελούν απειλή για τα Tears εκτός και αν τα δεύτερα προλάβουν να τα σκοτώσουν χτυπώντας τα με την μύτη τους.



εικόνα 23

5.3 Χειρισμός

Ο χειρισμός του παιχνιδιού είναι πολύ απλός και φαίνεται παρακάτω στην εικόνα 24.

εικόνα 24

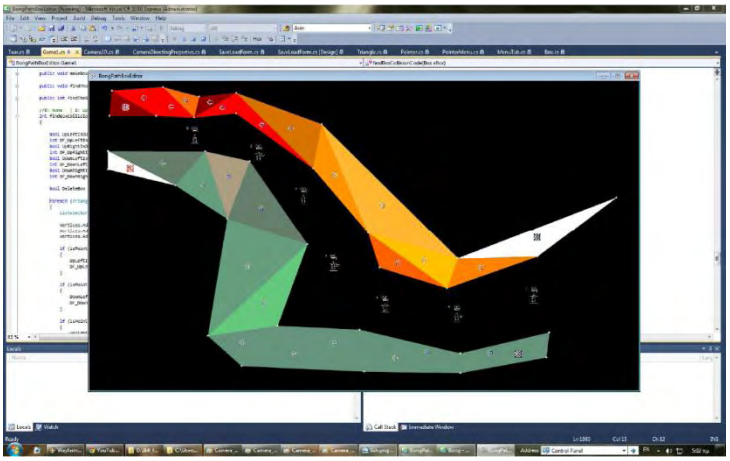
6. Παρουσίαση του Level Editor για το “Bong”

6.1 Η ιδέα και ο λόγος ύπαρξης του Editor

Ο Level Editor αυτού του παιχνιδιού υλοποιήθηκε σε C# χρησιμοποιώντας την βιβλιοθήκη της Microsoft για video games XNA 4.0, όπως και το ίδιο το παιχνίδι άλλωστε. Σκοπός της ύπαρξής του είναι η δημιουργία πιστών και η σκηνοθεσία αυτών, κάτι το οποίο θα μπορούσε να σκεφτεί κάποιος ότι μπορεί να γίνει και μέσα από τον πηγαίο κώδικα του παιχνιδιού και δεν θα είχε και άδικο. Δημιουργούνται δύο πολύ σημαντικά θέματα όμως με αυτό τον τρόπο. Πρώτον δεν θα πρέπει να θεωρείται δεδομένο ότι αυτός ο οποίος καλείται να σχεδιάσει τις πίστες του παιχνιδιού, ότι θα να γνωρίζει από προγραμματισμό και δεύτερον θα ήταν εξωφρενικά δύσκολο και πολύ κουραστικό κάθε φορά που θα άλλαζε ένα αντικείμενο της πίστας, να πρέπει να μεταγλωττίσουμε και να τρέξουμε το πρόγραμμά μας από την αρχή για να δούμε την αλλαγή. Είναι πιο εύκολο λοιπόν για των σχεδιαστή των πιστών να χρησιμοποιεί ένα πρόγραμμα το οποίο θα του δίνει την δυνατότητα να βλέπει τις αλλαγές που κάνει σε πραγματικό χρόνο και να σχεδιάζει χωρίς ιδιαίτερο κόπο αυτό που έχει φανταστεί.

Βασικό λοιπόν στοιχειό ενός παιχνιδιού είναι και ο game design editor. Είναι αυτός που θα αποτυπώσει στην οθόνη τις σκέψεις και την φαντασία του σεναριογράφου του εκάστοτε παιχνιδιού. Γι αυτό είναι πολύ σημαντικό να δοθεί ιδιαίτερη έμφαση στον σχεδιασμό και στην υλοποίηση του, έτσι ώστε να μπορούμε να έχουμε ένα αποτέλεσμα εφάμιλλο των προσδοκιών μας. Θα πρέπει να λάβουμε υπόψη μας τις τεχνικές που μπορούμε να χρησιμοποιήσουμε καθώς και τις δυνατότητες που μας προσφέρει η game engine που χρησιμοποιούμε. Ένα καλό level σε ένα παιχνίδι, δεν είναι μόνο τα γραφικά και οι κινήσεις που κάνει ο χαρακτήρας μας, αλλά και εάν υπάρχουν λάθη-bugs πάνω στον χάρτη που μπορούν να δημιουργήσουν τρύπες στο παιχνίδι, τα λεγόμενα cheats. Γι αυτό δεν πρέπει να δίνετε έμφαση μόνο στην αισθητική πλευρά αλλά και στην πρακτική, έτσι ώστε η αίσθηση που θα αφήνει στον παίκτη να είναι η καλύτερη δυνατή.

Στην εικόνα 25 δεξιά βλέπουμε ένα screenshot του level editor και μία πίστα που σχεδιάστηκε για το παιχνίδι Bong.



εικόνα 25

6.2 Τα κύρια στοιχεία Level Editor

Σε αυτό το κεφάλαιο λοιπόν αναλύονται πιο λεπτομερώς τα κύρια στοιχεία του παραπάνω level editor καθώς και οι δυνατότητες που παρέχονται στον σχεδιαστή. Οι λειτουργίες του είναι οι ακόλουθες:

Χειρισμός της Cameras (Viewport)

Το παράθυρο θέασης του παίχτη (Viewport) είναι σχεδιασμένο έτσι ώστε να ακολουθεί μία σειρά από αριθμημένους κόμβους. Η εισαγωγή ενός νέου κόμβου γίνεται με το κουμπί Y από το χειριστήριο, ενώ η θέση του επιλέγεται με το κουμπί A. Σε κάθε κόμβο δίνεται ένα μοναδικό ID προκειμένου να ταυτοποιείται και να ακολουθείται το επιθυμητό μονοπάτι, διατρέχοντας τους με αύξουσα σειρά. Κάθε κόμβος έχει τρεις μεταβλητές το Node ID που αναφέραμε πριν και το τις ακόλουθες δύο:

Zoom: Είναι το ποσοστό της εικόνας που βλέπει ο παίχτης όταν η camera βρίσκεται σε αυτό τον κόμβο.

Speed: Είναι η ταχύτητα που θα πρέπει να έχει η camera όταν βρεθεί σε αυτό τον κόμβο. Η ταχύτητα από κόμβο σε κόμβο μπορεί να διαφέρει, δίνοντας έτσι την αίσθηση της επιτάχυνσης ή της επιβράδυνσης αντίστοιχα.

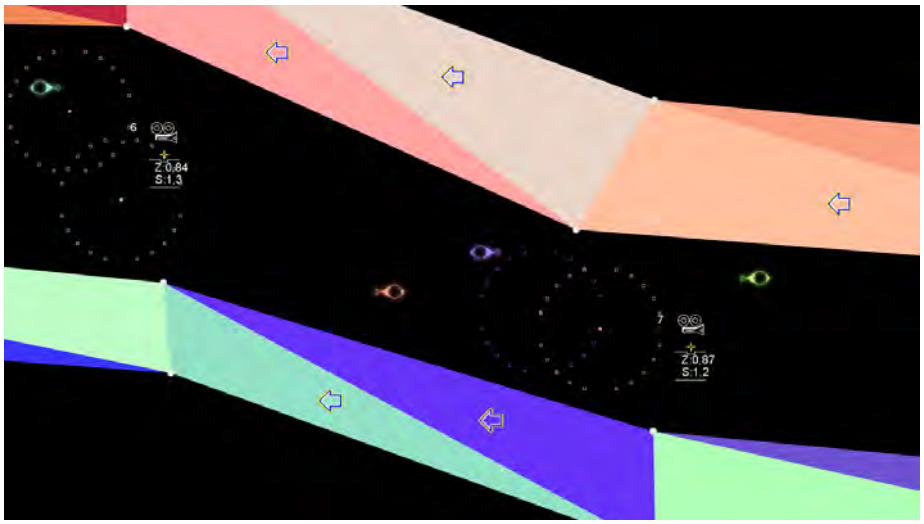
Τα παραπάνω χαρακτηριστικά έχουν ως αποτέλεσμα μίας πιο αρμονικής και κινηματογραφικής εμπειρίας.

εικόνα 26

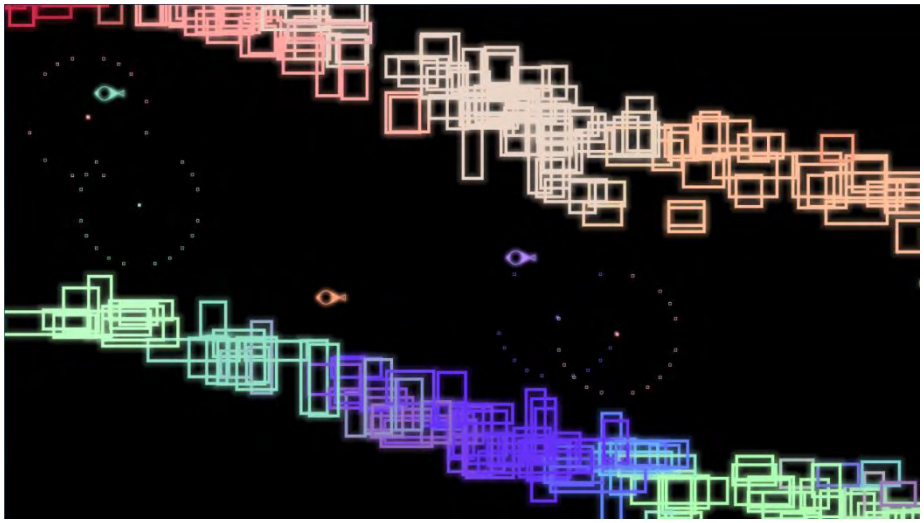
Στην εικόνα 26 βλέπουμε δύο κόμβους και τις τιμές που έχουμε δώσει στις μεταβλητές τους. Σε αυτή την περίπτωση η camera θέασης θα ξεκινήσει από τον κόμβο με ID 4, Zoom 1 και Speed 1,36 και γραμμικά θα καταλήξει στον κόμβο με ID 5, Zoom 1,23 και Speed 1,3.

Οριοθέτηση της πίστας (BoxFlow)

Στον χρήστη δίνεται η δυνατότητα μέσω του level editor να κατασκευάσει τρίγωνα ή τετράπλευρα μέσα στα οποία θα κινούνται τα "κουτιά". Τα τρίγωνα αυτά στη συνέχεια μπορούν να χρωματιστούν με οποιοδήποτε χρώμα, αλλάζοντας την RGB μεταβλητή τους. Για κάθε ομάδα τριγώνων που ορίζει ένα μονοπάτι, θα πρέπει να δημιουργηθεί μία ροή. Ο χρήστης το μόνο που πρέπει να κάνει είναι να ορίσει την αφετηρία από όπου θα ξεκινάνε τα κουτιά και να επιλέξει την επιλογή "Make BoxFlow". Αυτομάτως μέσα σε κάθε ομάδα τριγώνων θα δημιουργηθούν τα κουτιά μήτρες τα οποία είναι υπεύθυνα για την παραγωγή νέων κουτιών. Επίσης εκτελώντας την επιλογή "Make BoxFlow" κάθε τρίγωνο πλέον αποκτά ένα βέλος το οποίο δείχνει τι κατεύθυνση την οποία θα πρέπει να ακολουθήσει ένα κουτί αν γεννηθεί εκεί. Ο σχεδιαστής έχει πάντα τον τελευταίο λόγω ως προς την κατεύθυνση των βελών αφού ανά πάσα στιγμή μπορεί να την αλλάξει επιλέγοντας το συγκεκριμένο τρίγωνο και χρησιμοποιώντας το D-Pad. Τα κουτιά από την στιγμή που θα δημιουργηθούνε έρπονται από τρίγωνο σε τρίγωνο και αλλάζουν το χρώμα τους σύμφωνα με το χρώμα του τριγώνου στο οποίο βρίσκονται και αλλάζουν την πορεία τους όταν κάποιο γωνία τους βγει εκτός του μονοπατιού. Ακολουθούν δύο screenshots τα οποία δείχνουν πως κατασκευάζεται το BoxFlow στον editor (εικόνα 27) και πως αυτό δείχνει κατά την διάρκεια του παιχνιδιού (εικόνα 28).



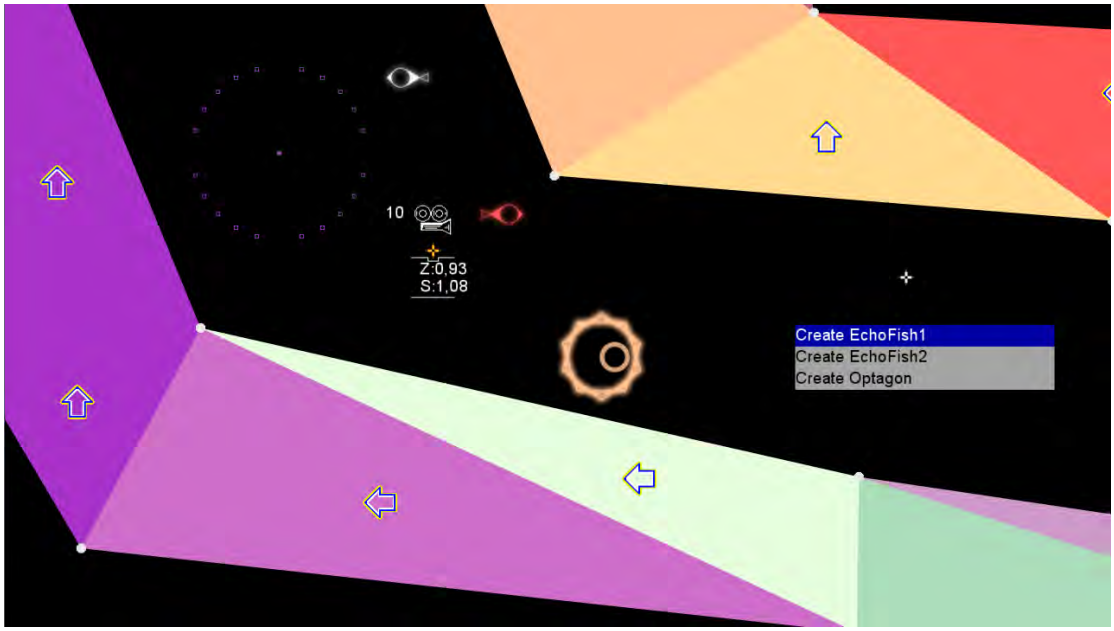
εικόνα 27



εικόνα 28

Εισαγωγή αντικειμένων

Χρησιμοποιώντας την κατάλληλη επιλογή από το μενού που εμφανίζεται πατώντας το πλήκτρο B στο controller, ο χρήστης μπορεί να εισάγει διάφορα αντικείμενα στην πίστα, όπως echofishes, optagones και biteNotes. Ο χρήστης μπορεί να αλλάξει το χρώμα κάθε αντικειμένου αλλάζοντας το αντίστοιχο RGB property, την θέση του καθώς και την κίνηση την οποία θέλει να εκτελεί το αντικείμενο. Επίσης στον σχεδιαστεί δίνεται η επιλογή να διαγράψει ένα αντικείμενο. Ένα στιγμιότυπο του level editor κατά την διάρκεια που εισάγουμε αντικείμενα μπορούμε να δούμε παρακάτω στην εικόνα 29.



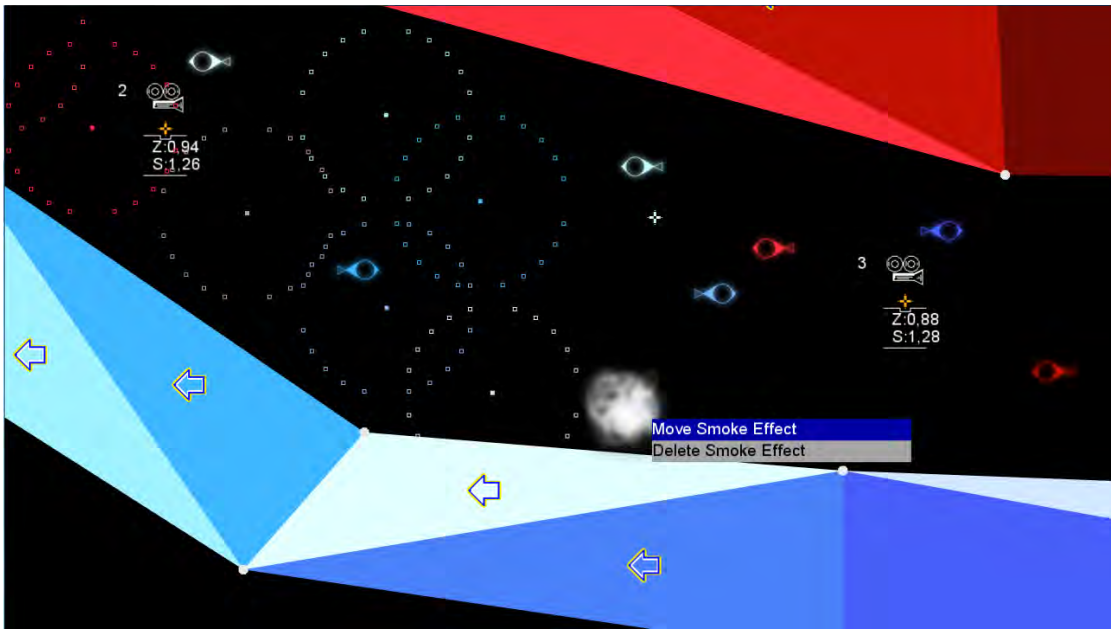
εικόνα 29

Εισαγωγή ειδικών εφέ (Special Effects)

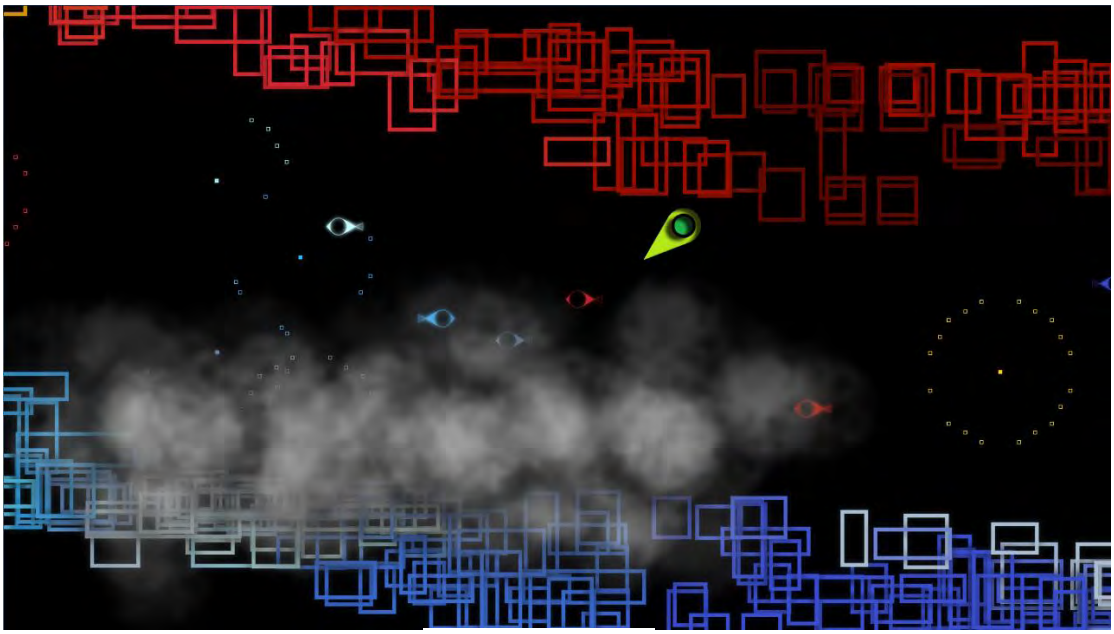
Αφού πρώτα κατασκευάσουμε το ειδικό εφέ που θέλουμε με το Mercury Particle Engine και το αποθηκεύσουμε σε μορφή XML μπορούμε πλέον να το περάσουμε στον level editor. Απαραίτητη προϋπόθεση για να μπορέσουμε να αναπαράγουμε το εφέ μας είναι πρώτα να έχουμε φορτώσει την βιβλιοθήκη του Mercury Particle Engine.

Πατώντας το πλήκτρο A και επιλέγοντας από το μενού “add smoke effect”, εμφανίζεται στην οθόνη μία εικόνα με ένα σύννεφο, το οποίο μπορούμε αν θέλουμε να το αλλάξουμε θέση ή να το σβήσουμε. Από αυτό το σύννεφο θα δημιουργηθεί αργότερα ένα εφέ καπνού το οποίο έχει ως στόχο να κρύψει είτε εχθρούς είτε κουτιά και γενικότερα να δυσκολέψει την ορατότητα του παίχτη. Στις παρακάτω εικόνες 30

– 31 βλέπουμε την διαδικασία εισαγωγής του ειδικού εφέ καθώς και το τελικό αποτέλεσμα που βλέπει στην οθόνη του ο χρήστης.



εικόνα 30



εικόνα 31

7. Απαιτήσεις Συστήματος - Εγκατάσταση

7.1 Απαιτήσεις Συστήματος - Εγκατάσταση

Για να εκτελέσουμε τόσο το παιχνίδι, όσο και τον level editor είναι απαραίτητο ο υπολογιστής μας να διαθέτει τα παρακάτω:

- Λειτουργικό Σύστημα Windows XP Service Pack 2, ή νεότερη έκδοση.
- Κάρτα γραφικών η οποία να υποστηρίζει το λιγότερο Shader Model 1.1.
- Χειριστήριο USB Microsoft Xbox 360.

Όσον αφορά την εγκατάσταση το μόνο που πρέπει να κάνουμε είναι να εκτελέσουμε το αρχείο "Bong.application" για το παιχνίδι και το αρχείο "BongPathBoxEditor.application" για τον level Editor. Μετά την εγκατάσταση τα ίδια αρχεία θα χρησιμοποιούμε και για την εκτέλεση των αντίστοιχων εφαρμογών.

8. Βιβλιογραφία

8.1 Βιβλιογραφία

- Koster, R. *A Theory of Fun for Game Design.* Paraglyph Press (2004).
- Mandana, S. *How to Design a Computer Game?* University of Washington, <http://courses.washington.edu/arch587/3.assignments/9.Final_Project/final-mandana.pdf>
- Jesse Liberty, Programming C#: Building .NET Applications with C#. O'Reilly Media - Fourth Edition (March 1, 2005)
- Aaron Reed, Learning XNA 4.0: Game Development for the PC, Xbox 360, and Windows Phone 7, O'Reilly Media (December 27, 2010)
- Jason Gregory, Game Engine Architecture, CRC Press - Second Edition (August 4, 2014)
- Bates Bob, Game Design 2nd Edition, Thomson Course Technology (2004)
- Brathwaite Brenda & Schreiber Ian, Challenges for Game Designers, Charles River Media (2009)
- Oxland Kevin, Gameplay and Design, Addison Wesley (2004)
- Matthew Davey, Mercury Particle Engine, <https://mpe.codeplex.com>