

ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΤΕ

**ΑΝΑΠΤΥΞΗ ΕΡΓΑΛΕΙΟΥ ΓΙΑ ΤΗΝ ΣΧΕΔΙΑΣΗ
ΠΑΙΧΝΙΔΙΩΝ ΜΕ ΤΗΝ ΒΟΗΘΕΙΑ ΥΠΟΛΟΓΙΣΤΗ**

**Πτυχιακή εργασία του
Γιώργου Μιχαηλίδη**

Επιβλέπων: Δρ. Νικόλαος Πεταλίδης. Επιστημονικός Συνεργάτης

ΣΕΠΡΕΣ, ΦΕΒΡΟΥΑΡΙΟΣ 2016

Υπεύθυνη δήλωση

Υπεύθυνη Δήλωση: Βεβαιώνω ότι είμαι συγγραφέας αυτής της πτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Μηχανικών Πληροφορικής ΤΕ του Τ.Ε.Ι. Κεντρικής Μακεδονίας.

Σύνοψη

Τα πρώτα ηλεκτρονικά παιχνίδια είχαν γραφτεί εξ' ολοκλήρου σε υλισμικό. Από τότε, οι κάρτες γραφικών και οι μικροεπεξεργαστές βελτιώθηκαν, δημιουργήθηκαν κονσόλες φτιαγμένες αποκλειστικά για ηλεκτρονικά παιχνίδια, με ειδικά χειριστήρια τα οποία σου προσφέρουν διαφορετικές εμπειρίες. Η διαδικασία ανάπτυξης λογισμικού είναι ακριβή και ο σχεδιασμός γίνεται όλο πιο σύνθετος και περίπλοκος. Τα έργα γίνονται όλο πιο απαιτητικά και δαπανηρά. Δημιουργήθηκε η ανάγκη για ένα εργαλείο το οποίο να παρέχει ένα ομοιογενές περιβάλλον για την ανάπτυξη σύνθετων έργων. Ένα CASE (Computer Aided Software Engineering) tool είναι ένα λογισμικό-εργαλείο το οποίο απλοποιεί τον κύκλο ανάπτυξης ενός λογισμικού. Στο τομέα του σχεδιασμού παιχνιδιών το πιο διαδεδομένο CASE tool είναι η μηχανή γραφικών. Μια μηχανή γραφικών είναι μια συνίτα από επαναχρησιμοποιήσιμα οπτικά εργαλεία τα οποία βρίσκονται σε ένα ενιαίο περιβάλλον. Σκοπός της πτυχιακής είναι να αναγνωριστούν μοτίβα και τεχνικές δημιουργίας παιχνιδιών, ώστε να δημιουργηθεί ένα εργαλείο το οποίο να το προσεγγίζει από υψηλό επίπεδο με αφαιρέσεις για εύκολη μοντελοποίηση και αυτοματοποίηση κατά τη δημιουργία.

Ευχαριστίες

Ευχαριστίες (στο μπαμπά, στη μαμά, κτλ)

Περιεχόμενα

| | |
|--|-----------|
| Υπεύθυνη δήλωση | 2 |
| Σύνοψη | 3 |
| Ευχαριστίες | 4 |
| 1 Εισαγωγή | 8 |
| 1.1 Δομή της εργασίας | 9 |
| 1.2 Εργαλεία και Frameworks | 9 |
| 1.3 Τεχνικές υλοποίησης | 11 |
| 1.4 Case Tools | 11 |
| 1.4.1 Κοινές λειτουργίες | 11 |
| 1.4.2 Τα πλεονεκτήματα του εργαλείου | 12 |
| 1.4.3 Χαρακτηριστικά ενός καλού case tool | 12 |
| 2 Ανάπτυξη ηλεκτρονικών παιχνιδιών | 14 |
| 2.1 Ανάπτυξη παιχνιδιών | 14 |
| 2.1.1 Μηχανές γραφικών | 14 |
| 2.1.2 Ιστορική αναδρομή | 15 |
| 2.1.3 Τι είναι μια μηχανή γραφικών | 15 |
| 2.1.4 Γιατί να χρησιμοποιήσει κάποιος μηχανη γραφικών; | 16 |
| 2.1.5 Δομή μιας μηχανής γραφικών | 16 |
| 2.2 Σχεδιασμός παιχνιδιών | 20 |
| 2.2.1 Μοντέλο MDA | 21 |
| 2.2.2 Πως ορίζεται ένα παιχνίδι | 21 |
| 2.3 Δομή μιας τυπικής ομάδας ανάπτυξης παιχνιδιών | 21 |

| | |
|--|-----------|
| 3 Ο πηρύνας της Μηχανής | 24 |
| 3.1 Κύκλος ζωής πηρύνα | 24 |
| 3.1.1 Βρόγχος παιχνιδιού | 25 |
| 3.1.2 Υποσυστήματα | 26 |
| 3.1.3 Τεχνικές Ενημέρωσης Υποσυστημάτων | 27 |
| 3.2 Διαχείριση οθονών | 27 |
| 3.2.1 Απαιτήσεις | 27 |
| 3.2.2 Συστατικά του συστήματος | 28 |
| 3.2.3 Αρχιτεκτονική | 30 |
| 3.2.4 Παράδειγματα χρήσης API | 31 |
| 3.3 Σύστημα εισόδων | 31 |
| 3.3.1 Human Interface devices | 31 |
| 3.3.2 Τύποι εισόδου | 32 |
| 3.3.3 Απαιτήσεις υποσυστήματος | 32 |
| 3.3.4 Observers-listeners | 32 |
| 3.4 Φυσική και συγκρούσεις | 35 |
| 3.4.1 Τα παιχνίδια ως soft real-time simulations | 35 |
| 3.4.2 Βασικές έννοιες φυσικής του συστήματος | 36 |
| 3.4.3 Οντότητες του συστήματος | 37 |
| 3.4.4 Αρχιτεκτονική | 38 |
| 3.5 Διεπαφή χρήστη | 38 |
| 3.5.1 Απαιτήσεις | 39 |
| 3.5.2 Δομή στη μνήμη | 40 |
| 3.5.3 Το σύστημα | 41 |
| 3.5.4 Τρόπος χρήσης | 43 |
| 3.6 Διαχείριση πόρων | 44 |
| 3.6.1 Ευθύνες του offline resource manager | 45 |
| 3.6.2 Ευθύνες του Runtime- Resource Manager | 45 |
| 3.7 Τεχνητή νοημοσύνη | 47 |
| 3.7.1 Δέντρα συμπεριφορών | 48 |
| 3.7.2 Finite State Machines | 50 |

| | |
|--|-----------|
| 4 Δικτύωση | 54 |
| 4.1 Το πρόβλημα | 54 |
| 4.1.1 Περιγραφή του προβλήματος | 54 |
| 4.1.2 Κατανόηση του προβλήματος | 54 |
| 4.1.3 Εξαγωγή απαιτήσεων | 55 |
| 4.2 Εκπόνηση σχεδίου | 56 |
| 4.2.1 Επιλογή πρωτοκόλου | 56 |
| 4.2.2 Επιλογή αρχιτεκτονικής δικτύου | 56 |
| 4.3 Σχεδίαση του framework | 57 |
| 4.3.1 Έννοιες | 57 |
| 4.3.2 Τύποι μηνυμάτων | 58 |
| 4.3.3 NetworkManager | 58 |
| 4.4 Υλοποίηση | 59 |
| 4.4.1 Τρόπος χρήσης | 59 |
| 4.4.2 Αρχιτεκτονική | 59 |
| 4.4.3 Packages Module | 59 |
| 4.4.4 Networking Module | 61 |
| 4.4.5 API | 64 |
| 4.5 Testing | 67 |
| 5 Διαγνωστικά και Αποσφαλμάτωση | 69 |
| 5.1 Logging and Tracing | 69 |
| 5.2 Time Benchmarking | 70 |
| 5.3 Debug Drawing API | 70 |
| 5.4 Υποσυστήματα αποσφαλμάτωσης | 71 |
| 6 Επίλογος | 73 |
| 6.1 Επεκτασιμότητα | 74 |
| 6.1.1 Plugins | 74 |
| 6.2 Συμπεράματα | 74 |

Κεφάλαιο 1

Εισαγωγή

Σκοπός Σκοπός της πτυχιακής είναι να εξηγήσει σε υψηλό επίπεδο τη σύνδεση και αλληλεπίδραση βασικών υποσυστήματων τα οποία απαρτίζουν μια τυπική μηχανή γραφικών, μαζί με διαγράμματα και παραδείγματα χρήσης του API. Η δομή των μηχανών γραφικών στα κατώτερα επίπεδα είναι πανομοιότυπη. Οι μεγάλες διαφορές των μηχανών γραφικών βρίσκονται στην υλοποίηση. Πρακτικά, οι μηχανές γραφικών απαρτίζονται από τα ίδια υποσυστήματα. Πολλά βιβλία έχουν γραφτεί τα οποία εστιάζουν στο κάθε υποσύστημα ξεχωριστά, όπως π.χ. την απόδοση (rendering). Η πτυχιακή εστιάζει στην αρχιτεκτονική των υποσυστημάτων, για το πώς τα υποσυστήματα είναι οργανωμένα, ποιες λειτουργίες και μοτίβα επαναλαμβάνονται κατά τη δημιουργία παιχνιδιών, ποιες είναι οι απαιτήσεις για το κάθε μεγάλο υποσύστημα της μηχανής και πώς οργανώνεται ένα υποσύστημα χωρίς να έχει δεσμεύσεις με συγκεκριμένη πλατφόρμα και εξαρτήσεις από άλλα υποσυστήματα.

Υλοποίηση Η υλοποίηση χωρίζεται σε δύο μεγάλα μέρη.

- Gem Engine η βιβλιοθήκη η οποία περιέχει τα κύρια υποσυστήματα που απαρτίζουν τη μηχανή γραφικών
- Gem IDE το γραφικό περιβάλλον της μηχανής το οποίο προσφέρει οπτική αναπαράσταση των λειτουργιών του Gem Engine.

Η υλοποίηση αξιοποιεί object-oriented και functional paradigms και είναι γραμμένη σε C# και κατά την ανάπτυξη του πηγαίου κώδικα χρησιμοποιήθηκε το σύστημα ελέγχου

αναθεωρήσεων git. Ο πηγαίος κώδικας βρίσκεται στο [4] και [6] και η τεκμηρίωσή του στο [5].

1.1 Δομή της εργασίας

1. Ανάπτυξη παιχνιδιών: Επεξήγηση των εννοιών και διαφορών της ανάπτυξης παιχνιδιών (game development) και του σχεδιασμού παιχνιδιών (game design). Ανάλυση μιας τυπικής ομάδας ανάπτυξης παιχνιδιών, στην οποία απευθύνεται το εργαλείο και οριοθέτηση του τι θεωρείται παιχνίδι.
2. Ο πηρύνας της μηχανής: Ανάλυση βασικών υποσυστημάτων τα οποία βρίσκονται στον πηρύνα της βιβλιοθήκης και είναι αναγκαία για τη λειτουργία της μηχανής.
3. Δικτύωση: Περιγραφή του υποσυστήματος δικτύωσης της μηχανής.
4. Διαγνωστικά και αποσφαλμάτωση: Τεχνικές ελέγχου υγείας του λογισμικού τόσο κατά τη διαδικασία δημιουργίας όσο και στις τελικές εκδόσεις.
5. Επίλογος: Συμπεράσματα και τεχνικές επέκτασης.

1.2 Εργαλεία και Frameworks

Με την διόγκωση των βιβλιοθηκών ανοιχτού λογισμικού, έγινε εύκολη η εύρεση και ενσωμάτωση κώδικα τρίτων. Σε ιστοσελίδες φιλοξενίας λογισμικού ανοικτού κώδικα υπάρχουν αποδεκτές και πολυχρησιμοποιημένες λύσεις για πολλά προβλήματα ώστε να μην χρειάζεται να ξαναεφεύρεις τον τρόχο. Η πτυχιακή χρησιμοποιεί διάφορα εργαλεία και βιβλιοθήκες για να μπορεί να επικεντρωθεί σε άλλα επίπεδα. Τα βασικά εργαλεία τα οποία χρησιμοποιεί ή επεκτίνει είναι τα παρακάτω:

- Monogame Βιβλιοθήκη για απόδοση σε πολλές πλατφόρμες (cross-platform rendering) η οποία βρίσκεται ένα επίπεδο πάνω από τα drivers της κάρτας γραφικών.
- SDL Simple DirectMedia Layer μία cross-platform development library. Χρησιμοποιείται για το σύστημα ήχου και εισόδου.
- TPL Ασύγχρονα πρότυπα για ταυτόχρονο και παράλληλο προγραμματισμό.

- Lidgren Αφαίρεση για networks socket. Χρησιμοποιείται από το υποσύστημα διαδικτύωσης.
- Castle Proxy Βιβλιοθήκη για τροποποίηση κώδικα σε πιο χαμηλά επίπεδα (intermediate language).
- Farseer Βιβλιοθήκη φυσικής. Χρησιμοποιείται από το υποσύστημα φυσικής και εντοπισμού συγκρούσεων.
- Autofac IOC container. Το κάθε υποσύστημα δημοσιεύει τις αφαιρέσεις του μέσα από το container. Χρησιμοποιείται για σκοπούς επεκτασιμότητας των υλοποιήσεων, ομαδοποίησης λειτουργιών, δυναμικής αλλαγής συμπεριφορών και για την εύρεση εξαρτήσεων
- NLog Logger. Το προεπιλεγμένο framework στο Auditing του υποσυστήματος διαγνωστικών.
- Glsl Shaders. OpenGL Shading Language η οποία χρησιμοποιείται κατευθείαν στο pipeline της κάρτας γραφικών. Χρησιμοποιείται για καλή επίδοση στις σκιές, ανακλάσεις, εφέ και σε υπολογισμούς οι οποίοι είναι πολύ εντατικοί για τον επεξεργαστή.
- MEF Managent Extensibility Framework. Βιβλιοθήκη για δημιουργία λογισμικού με δυνατότητες δυναμικής επέκτασης κα διαχείρισης plugins. Χρησιμοποιείται για να βρίσκει και να φορτώνει τα plugins του περιβάλλοντος ανάπτυξης.
- WPF Windows Presentation Foundations. Χρησιμοποιείται για τη δημιουργία του γραφικού περιβάλλοντος του περιβάλλοντος ανάπτυξης.
- Roslyn C# compiler ανοιχτού κώδικα με δυνατότητα scripting
- Rx Observable streams για προγραμματισμό οδηγούμενο από συμβάντα.
- Caliburn Model–view–viewmodel για WPF. Το MVVM διαχωρίζει το γραφικό περιβάλλον από τη λογική ενημέρωσής του.
- Moq Χρησιμοποιείται για δημιουργία ψεύτικων αντικειμένων με τροποποίηση συμπεριφορά για δοκιμές.

- MsTest Εργαλείο δοκιμών μονάδας βασισμένο σε ισχυρισμούς.
- VS 2015 Ultimate Ολοκληρωμένο περιβάλλον ανάπτυξης για λογισμικό με ενσωματωμένο C# profiler για ανάλυση των επιδόσεων και επιθεώρηση της κατανομής μνήμης.

1.3 Τεχνικές υλοποίησης

Η υλοποίηση βασίστηκε αυστήρα σε SOLID principles και functional / object oriented σχεδιαστικά μοτίβα. Οι κλάσεις δεν έχουν εξαρτήσεις σε υλοποιήσεις και είναι immutable, δηλαδή μετά τη δημιουργία τους δεν μπορεί να αλλάξει η εσωτερική τους κατάσταση (internal state), για αποφυγή ανεπιθύμητων συμπεριφορών. Το API είναι declarative, fluent και βασισμένο σε αφαιρέσεις με εύκολα τροποποιήσιμο bootstrapping λόγω inversion of control.

1.4 Case Tools

Η διαδικασία ανάπτυξης λογισμικού είναι ακριβή και ο σχεδιασμός γίνεται όλο πιο σύνθετος και περίπλοκος. Τα έργα γίνονται όλο πιο απαιτητικά και δαπανηρά. Δημιουργήθηκε η ανάγκη για ένα εργαλείο το οποίο να παρέχει ένα ομοιογενές περιβάλλον για την ανάπτυξη σύνθετων έργων. Ένα CASE (Computer Aided Software Engineering) tool είναι ένα λογισμικό-εργαλείο το οποίο απλοποιεί τον κύκλο ανάπτυξης ενός λογισμικού. Τα Case Tools γίνονται όλο και πιο δημοφιλές, λόγω της βελτίωσης των δυνατοτήτων και της λειτουργικότητας στην ανάπτυξη της ποιότητας του λογισμικού. Η διαδικασία ανάπτυξης αυτοματοποιείται, και συντονίζεται. Το λογισμικό συντηρείται και αναλύεται εύκολα.

1.4.1 Κοινές λειτουργίες

Η ντετερμινιστική οριοθέτηση των λειτουργίες ενός case tool δεν μπορεί να γίνει γιατί οι λειτουργίες διαφέρουν ανάλογα με το σκοπό χρήσης. Οι λειτοργίες οι οποίες επαναλαμβάνονται είναι οι παρακάτω:

- Δημιουργία ροής δεδομένων και μοντέλων οντοτήτων.

- Καθιέρωση της σχέσης μεταξύ απαιτήσεων και προτύπων.
- Ανάπτυξη του σχεδιασμού σε υψηλό επίπεδο.
- Ανάπτυξη λειτουργικών και διαδικαστικών περιγραφών
- Ανάπτυξη περιπτώσεων δοκιμών (test cases).

1.4.2 Τα πλεονεκτήματα του εργαλείου

Η ανάπτυξη λογισμικού με CASE tools έχει πολλά πλεονεκτήματα. Η εγκατάσταση του είναι γρήγορη και εξοικονομείται χρόνος μειώνοντας τον χρόνο στον προγραμματισμό και τις δοκιμές, αφού πολλές λειτουργίες είναι αυτοματοποιημένες. Υπάρχει η δυνατότητα οπτικοποίησης του κώδικας και της ροή δεδομένον για καλύτερη κατανόηση και ελέγχου του λογισμικού. Το εργαλείο αναλαμβάνει για την βέλτιση χρήση των διαθέσιμων πόρων ούτως ώστε η ομάδα ανάπτυξης να επικεντρωθεί στον πυρήνα της εφαρμογής τους. Η ανάλυση ανάπτυξη και σχεδιασμός γίνεται με ενιαίες μεθοδολογίες και η τεκμηρίωση δημιουργείται και τροποποιείται αυτόματα. Πολλά εργαλεία περιέχουν εξαγωγή των δεδομένων σε γνωστές μορφές (well known formats) για μεταφορά πληροφοριών ανάμεσα σε διάφορα εργαλεία. Σε μια βιομηχανία όπου ο χρόνος είναι ασφυκτικός, οι απαιτήσεις αλλάζουν συνέχεια και τα εκδόσεις λογισμικών γίνονται όλο και πιο συχνές, η αυτοματοποίηση των διαφόρων δραστηριοτήτων των διαδικασιών ανάπτυξης και διαχείρισης του συστήματος αυξάνει την παραγωγικότητα της ομάδας ανάπτυξης

1.4.3 Χαρακτηριστικά ενός καλού case tool

Ένα case tool θεωρείται καλό και χρήσιμο αν έχει τα παρακάτω χαρακτηριστικά.

- Τυποποιημένη μεθοδολογία χρησιμοποιώντας τεχνικές μοντελοποίησης όπως UML.
- Flexibility: το εργαλείο πρέπει να προσφέρει τη δυνατότητα στο χρήστη να επιλέγει ποια εργαλεία να χρησιμοποιήσει.
- Strong integration: το εργαλείο πρέπει να υποστηρίζει όλα τα στάδια ανάπτυξης. Όταν γίνεται μια αλλαγή, τα στάδια τα οποία επιρεάζονται πρέπει να τροποποιούνται κατάλληλα.

- Ενσωμάτωση με εργαλεία ελέγχου.
- Reverse-engineering: δυνατότητα δημιουργίας κώδικα από δεδομένα

Κεφάλαιο 2

Ανάπτυξη ηλεκτρονικών παιχνιδιών

Ανάπτυξη ηλεκτρονικών παιχνιδιών (game development) ονομάζουμε τη διαδικασία της δημιουργίας ενός παιχνιδιού. Η ομάδα ανάπτυξης μπορεί να κυμαίνεται από ένα άτομο μέχρι μια μεγάλη επιχείρηση.

2.1 Ανάπτυξη παιχνιδιών

Η ανάπτυξη παιχνιδιών περιλαμβάνει πολλές εξειδικευμένες ειδικότητες, οι οποίες παράγουν μια μηχανή παιχνιδιών.

2.1.1 Μηχανές γραφικών

Στο τομέα του σχεδιασμού παιχνιδιών το πιο διαδεδομένο CASE tool είναι η μηχανή γραφικών. Μια μηχανή γραφικών είναι μια σουίτα από επαναχρησιμοποιήσιμα οπτικά εργαλεία τα οποία βρίσκονται σε ένα ενιαίο περιβάλλον. Η κεντρική λειτουργικότητα η οποία παρέχεται περιλαμβάνει τη φωτοαπόδοση σε πραγματικό χρόνο (real time rendering), τη μηχανή φυσικής και εντοπισμό συγκρούσεων (physics and collision detection), το scripting, το animation, την τεχνητή νοημοσύνη (Artificial Intelligence), τη δικτύωση (networking), τον παραλληλισμό ενεργειών (multitasking), την διαχείριση μνήμης και τον γράφο σκηνής (scene graph). Η ανάπτυξη των παιχνιδιών μέσω μιας μηχανής γραφικών γίνεται εύκολα, γρήγορα και οδηγούμενη από δεδομένα (data driven) ούτως ώστε οι δημιουργοί παιχνιδιών να μπορούν να ασχολούνται με τις λεπτομέρειες του παιχνιδιού τους. Οι μηχανές αναπτύσσονται από ομάδες που απαρτίζονται όχι μόνο

από προγραμματιστές, αλλά και από μαθηματικούς, φυσικούς κλπ. Η κάθε υπο-ομάδα εστιάζει σε ένα συγκεκριμένο κομμάτι, όπως οι φυσικοί με τον εντοπισμό συγκρούσεων, και αρχιτέκτονες λογισμικού σχεδιάζουν το πως τα κομμάτια συνδέονται και αλληλεπιδρούν μεταξύ τους, χωρίς να τους απασχολούν οι λεπτομέριες σχεδίασης του κάθε κομματιού.

2.1.2 Ιστορική αναδρομή

Η ιστορία των Βιντεοπαιχνιδιών, αρχίζει στα τέλη της δεκαετίας του '40. Προς τα τέλη του '50 και στα μέσα του '60, στην Αμερική, αρχίζουν να μπαίνουν στην καθημερινή μας ζωή, οι υπολογιστές. Για την ακρίβεια, οι κεντρικοί υπολογιστές. Από εκείνη την περίοδο, τα βιντεοπαιχνίδια έκαναν την εμφάνιση τους, στις κονσόλες, στα φλίπερ, στους υπολογιστές, αλλά και στις φορητές κονσόλες. Από τότε η δημιουργία παιχνιδιών έχει γιγαντιωθεί έχοντας ένα τεράστιο κομμάτι της παγκόσμιας οικονομίας. Πλέον ο ανταγωνισμός είναι τεράστιος, τα βιντεοπαιχνίδια κυκλοφορούν για διάφορες κονσόλες με πολύ απαιτητικά γραφικά και με πολύ γρήγορο ρυθμό.

2.1.3 Τι είναι μια μηχανή γραφικών

Η πρώτη αναφορά σε μηχανή γραφικών έγινε στα μέσα της δεκαετίας του 90 και αναφερόταν στο δημοφιλές παιχνίδι Doom του οποίου η αρχιτεκτονική διαχώριζε τα βασικά συστήματα του παιχνιδιού, όπως rendering system, collision detection system, audio system, asset system κλπ. Η αξία αυτού του διαχωρισμού εκτιμήθηκε από την κοινότητα όταν οι προγραμματιστές ξεκίνησαν να πουλάνε άδειες για το λογισμικό, επαναχρησιμοποιούσαν εργαλεία προηγούμενων παιχνιδιών με δημιουργία νέων assets. Μικρότερα στούντιο τροποποιούσαν εκδόσεις υπάρχων παιχνιδιών χρησιμοποιώντας το SDK.ό Πολλά παιχνίδια γράφτηκαν με σκοπό να επαναχρησιμοποιηθούν κομμάτια κόδικα και modding. Πολλές μηχανές όπως η μηχανή του Quake III γράφτηκαν με τρόπο ώστε να είναι εύκολα προσαρμόσημες χρησιμοποιώντας scripting, με σκοπό την εμπορευματοποίηση μέσω licensing. Η διαχωριστική γραμμή μεταξύ του παιχνιδιού και της μηχανής δεν μπορεί να οριστεί με ακρίβεια. Πολλές μηχανές μπορεί να περιέχουν συγκεκριμένα μέρη που αφορούν συγκεκριμένη λειτουργία του παιχνιδιού. Η μεγάλη διαφορά είναι στο data-driven architecture όπου οι κανόνες και τα στοι-

χεία δεν είναι hard-coded αλλά διαβάζονται από εξωτερικό αρχείο. Οι μηχανές έχουν τα όριά τους ανάλογα με τα είδη παιχνιδιού στα οποία η μηχανή εστιάζει, σε ποιες πλατφόρμες, σε τι στιλ γραφικών, σε ποια αρχιτεκτονική της γρυ πλπ.

2.1.4 Γιατί να χρησιμοποιήσει κάποιος μηχανή γραφικών;

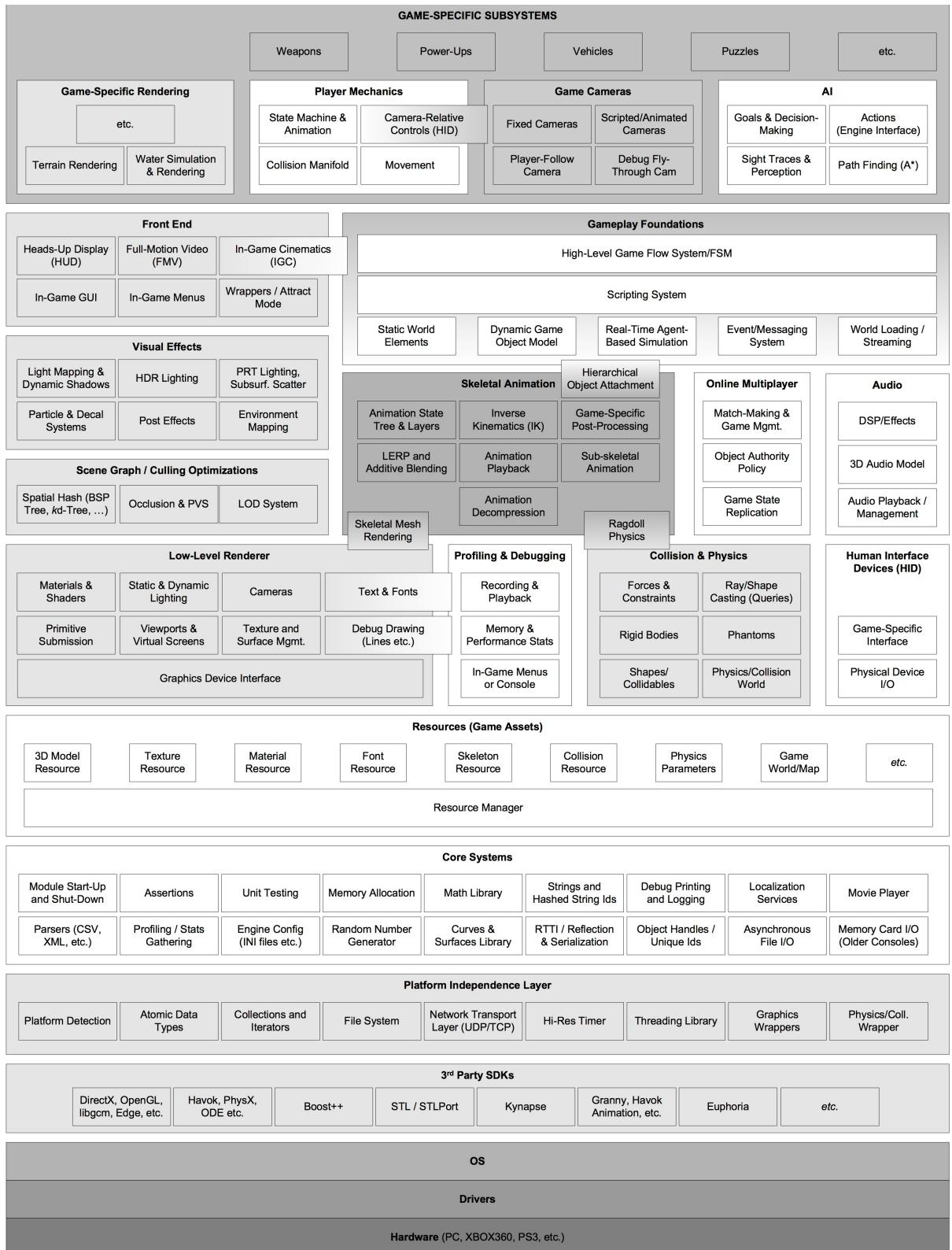
Η αφαίρεση πάντα βοηθούσε τον εγκέφαλο να λειτουργήσει καλύτερα και να κατανοήσει αλληλεπιδράσεις μεταξύ συστημάτων και περίπλοκες έννοιες. Οι μηχανές γραφικών απαλλάζουν τους γραφίστες και τους προγραμματιστές από τις τεχνικές λεπτομέριες, και εστιάζουν στην αισθητική και στο gameplay. Επίσης με την αποσύνδεση των συστημάτων έχουμε πιο προβλέψημη συμπεριφορά, επεκταστημότητα των υποσυστημάτων ως υποσυστήματα και ευκολη δοκιμαστικότητα.

2.1.5 Δομή μιας μηχανής γραφικών

Η μοντέλοποίηση της δομής των μηχανών γραφικών σε αφαιρετικό επίπεδο είναι πανομοιότυπη, διαφέρει όμως η υλοποίηση. Η δομή ξεκινά να διαφοροποιείται στα υψηλότερα επίπεδα, τα οποία υλοποιούνται με στόχο για σχεδιασμό συγκεκριμένου είδους παιχνιδιών. Μια τυπική μηχανή γραφικών παρουσιάζεται στο 2.1 [2].

Επισκόπηση επιπέδων

- Hardware Το hardware αντιπροσωπεύει το υλικό, δηλαδή το υπολογιστή ή κονσόλα στο οποίο θα τρέξει το παιχνίδι.
- Drivers Τα drivers διαχειρίζονται τους πόρους του υλικού και του λειτουργικού ένα καθολικό προτόκολο επικοινωνίας μεταξύ των πολλών παραλλαγών.
- OS (Operating System) Το λειτουργικό σύστημα είναι ο ενορχηστρωτής των προγραμμάτων. Κατανέμει το χρόνο μεταξύ των πόρων του υλικού και των προγραμμάτων και παραλληλίζει την εκτέλεση προγραμμάτων. Στις κονσόλες το λειτουργικό παίζει τυπικό ρόλο, αφού το λογισμικό (παιχνίδια) έχουν σχεδόν πλήρη έλεγχο του υλικού. Πλέον όμως στις σύγχρονες κονσόλες και στα λειτουργικά συστήματα όπως Microsoft Windows, κανένα πρόγραμμα δεν έχει πλήρη έλεγχο, αφού το λειτουργικό μπορεί να μοιραστεί πόρους του συστήματος με το τρέχον λογισμικό για να εμφανίσει για παράδειγμα κάποιο μήνυμα.



Διάγραμμα 2.1: Τυπική αρχιτεκτονική μηχανής γραφικών

- Third party SDKs and MiddleWare Πολλές φορές χρησιμοποιούνται βιβλιοθήκες ανάπτυξης λογισμικού τρίτων, οι οποίες γράφτηκαν από την κοινότητα και λύνουν κοινά προβλήματα. Συχνό παράδειγμα είναι η χρήση βιβλιοθηκών για κοινές δομές δεδομένων, η για rendering.
- Collision and Physics Εντοπισμός συγκρούσεων και rigid-body dynamics (physics) δηλαδή πως ένα σύστημα με διασυνδεδεμένα σώματα αντιδρά όταν του ασκούνται εξωτερικές δυνάμεις.
- Platform Independence Layer Πολλά παιχνίδια καλούνται να τρέξουν σε περισσότερες από μία πλατφόρμες. Σε αυτό το επίπεδο ενθυλακώνονται οι εντολές στο λειτουργικό σύστημα και στο υλικό για να μπορούν να αλλάξουν ανάλογα με το περιβάλλον ανάπτυξης.
- Core Systems Κάθε μεγάλο και σύνθετο πρόγραμμα χρειάζεται κάποιες γενικής χρήσης βιβλιοθήκες. Κάποιες από αυτές είναι:
 - Assertions κώδικας ο οποίος ελέγχει υποθέσεις του προγραμματιστή για το πως θα τρέξει η συνάρτηση. Ο κώδικας αυτός αφαιρείται στις τελικές εκδόσεις, αφού περάσει η περίοδος δοκιμών.
 - Memory Management: έλεγχος της μνήμης για αποφυγή fragmentation και out of memory.
 - Βιβλιοθήκη μαθηματικών η οποία περιέχει μαθηματικά που εμφανίζονται στις προσομειώσεις πραγματικού χρόνου, όπως μαθηματικά διανυσμάτων, πινάκων, γεωμετριάς κλπ.
- Resource Management διαχείριση των πόρων του παιχνιδιού, όπως τα μοντέλα, τα textures, ο κόσμος, οι χάρτες κλπ
- Rendering Engine το πιο σύνθετο κομμάτι, το οποίο είναι υπεύθυνο για την αναπαράσταση του παιχνιδιού στην οθόνη.
- Low-level renderer το οποίο εστιάζει στην βελτιστοποίηση των πρωτογενή γεωμετρικά σχήματα που απαρτίζουν τα διάφορα κομμάτια.

- Scene Graph η οποία καθορίζει ποια αντικείμενα πρέπει να γίνουν render από τη rendering engine χρησιμοποιώντας αλγόριθμους ανάλογα με το μέγεθος και το είδος του παιχνιδιού.
- Visual Effects που αποτελούνται από particle systems, light mapping, dynamic shadows, anti-aliasing
- Front End πολλές φορές χρειάζεται κάποιο επίπεδο επικοινωνίας με το παιχνίδι, όπως μενού κονσόλα, pop-ups κλπ
- Profiling and Debugging. Η ποιότητα της απόδοσης του παιχνιδιού είναι πολύ κρίσιμη. Για να γίνει βελτιστοποίηση της απόδοσης, χρειάζονται εργαλεία ανάλυσης του υλικού ενώ τρέχει το παιχνίδι με οπτική αναπαράσταση.
- Collision and Physics Η φυσική και οι συγκρούσεις είναι στενά συνδεδεμένες γιατί οι συγκρούσεις επιλυόνται με κανόνες φυσικής.
- Animations
- Human Interface Devices Βιβλιοθήκες για έλεγχο των σημάτων από το πληκτρολόγιο, το ποντίκι, τα χειριστήρια τα οποία χρησιμοποιεί ο χρήστης για να επικοινωνήσει με το παιχνίδι. Επίσης πολλές φορές χρησιμοποιούνται για να επικοινωνήσει το παιχνίδι με το χρήστη, όπως η δόνηση στο χειριστήριο.
- Audio Το σύστημα διαχείρισης ήχου έχει πολλές δυνατότητες. Μπορεί να χρησιμοποιηθεί για τρισδιάστατη αναπαράσταση του ήχου, για να δώσει την αίσθηση του βάθους και της απόστασης στον χρήστη.
- Online Multiplayer – Networking Συνήθως ένας χρήστης παίζει μόνος του σε ένα εικονικό κόσμο. Πολλές φορές όμως άλλοι χρήστες συνδέονται σε αυτό τον κόσμο.
- Gameplay Foundation Systems οι κανόνες του εικονικού κόσμου οι οποίοι οριοθετούν τις δυνατότητες του παίχτη Game Worlds and Object Models η αναπαράσταση του κόσμου με αντικειμενοστραφή τρόπο, το game object model. Περιλαμβάνει το στατικό background, τα dynamic rigid bodies, τους παίχτες (Player Characters), τους non-player characters, τα όπλα, τα οχήματα, το φωτισμό, τις κάμερες κλπ.

- Event System το σύστημα που επιτρέπει την επικοινωνία μεταξύ του game object model.
- Scripting System Το scripting σε μία μηχανή γραφικών, επιταχύνει την ανάπτυξη γιατί περιέχει χρήσιμες, συχνές και έτοιμες προς εκτελεση εντολές.
- Artificial Intelligence Τυπικά patterns νοημοσύνης, όπως navigation , path finding, dynamic object avoidance κλπ.
- Game Specific Subsystems Τα συστήματα που είναι χτισμένα πάνω από τη μηχανή γραφικών και αφορούν συγκεκριμένα το παιχνίδι.
- Tools and Asset Pipeline Όλα τα παιχνίδια χρειάζονται πολλά δεδομένα για να αναπαραστήσουν τον εικονικό κόσμο, όπως configuration, scripts, τρισδιάστατα μοντέλα κλπ. Οι μηχανές πρέπει να είναι σε θέση να επεξεργάζονται συγκεκριμένου τύπου δεδομένα τα οποία εξάγονται από δημοφιλή digital content creation προγράμματα.
- Resource Database Λόγω των πολλών δεδομένων, οι μηχανές πρέπει να υλοποιούν τεχνικές αναζήτησης και αποθήκευσης. Μπορούν να χρησιμοποιηθούν από υπάρχων βάσεις δεδομένων, μέχρι xml αρχεία.

2.2 Σχεδιασμός παιχνιδιών

Σχεδιασμό παιχνιδιών (game design) ονομάζουμε τον σχεδιασμό και την εφαρμογή τεχνικών αισθητικής στη δημιουργία ενός παιχνιδιού με σκοπό τη διευκόλυνση της αλληλεπίδρασης μεταξύ των παικτών. Οι μηχανές γραφικών χρησιμοποιούνται για σκοπούς game design. Οι game designers σχεδιάζουν το πώς θα είναι το παιχνίδι χωρίς να τους απασχολούν οι τεχνικές λεπτομέριες. Είναι υπεύθυνοι για:

- Τα εργαλεία και μηχανισμούς μέσα στο παιχνίδι
- Για την ανάπτυξη κανόνων
- Ιστορία και πλοκή
- Στρατηγική και τυχαιότητα

Όπως και το game engineering, το game design χωρίζεται σε υποκατηγορίες και τεχνικές. Οι κατηγορίες αυτές διαφέρουν ανάλογα με το μέγεθος της ομάδας και το παιχνίδι το οποίο σχεδιάζεται.

2.2.1 Μοντέλο MDA

Στο μοντέλο MDA ο σχεδιασμός παιχνιδιών χωρίζεται στα mechanics, dynamics και aesthetics. [7].

- Mechanics: τα διάφορα συστατικά ενός παιχνιδιού, στο επίπεδο αλγορίθμων και της αναπαράστασης.
- Dynamics: η συμπεριφορά των mechanics κατά των χρόνο εκτέλεσης, αντιδρώντας στις εισόδους και εξόδους του συστήματος με την πάροδο του χρόνου
- Aesthetics: οι επιθυμητές συναισθηματικές αντιδράσεις τις οποίες προκαλεί η συσκευή αναπαραγωγής όταν αλληλεπιδρά με τον παίχτη.

2.2.2 Πως ορίζεται ένα παιχνίδι

Διαισθητικά ο καθένας μπορεί να ξεχωρίσει τι είναι ένα παιχνίδι όπως το σκάκι και η μονόπολη. Στη θεωρία παιχνιδιών, ένα παιχνίδι είναι ένα σύνολο παραγώντων οι οποίοι δρουν με βάση στρατηγικών και τεχνικών με στόχο να μεγιστοποιήσουν τα κέρδη μέσα στον εικονικό κόσμο μέσα σε ένα πλαίσιο καλά ορισμένων κανόνων.

Ο Raph Koster στο βιβλίο του a Theory of Fun for Game Design [3], ορίζει το παιχνίδι ως μια διαδραστική εμπειρία η οποία παρέχει στο χρήστη μια προκλητική σειρά από patterns τα οποία μαθαίνει και στην τελική εξειδικεύεται. Ισχυρίζεται ότι η εκμάθηση και εξειδίκευση βρίσκονται στην καρδιά στο τι θεωρείται διασκεδαστικό, όπως ένα λογοπαίγνιο γίνεται αστείο τη στιγμή που ο εγκέφαλος αντιληφθεί το pattern.

2.3 Δομή μιας τυπικής ομάδας ανάπτυξης παιχνιδιών

Πριν από την ανάλυση της δομής της μηχανής, θα γίνει ανάλυση της δομής ομάδας η οποία θα την χρησιμοποιεί για να αναπτυχθούν στοχευμένα εργαλεία για το κάθε πρόβλημα της κάθε υπο-ομάδας.

Μηχανικοί Οι μηχανικοί σχεδιάζουν και υλοποιούν το λογισμικό του παιχνιδιού και τα εργαλεία τα οποία χρησιμοποιούνται για την ανάπτυξή του. Οι δύο μεγάλες κατηγορίες μηχανικών είναι οι

- runtime programmers οι οποίοι ασχολούνται με τη μηχανή κται το παιχνίδι
- tool programmers οι οποίοι γράφουν tools τα οποία αυτοματοποιούν και ευκολύνουν την διαδικασία ανάπτυξης.

Οι μηχανικοί έχουν είτε κάποια ειδικότητα, για παράδειγμα ειδικότητα στη τεχνητή νοημοσύνη, είναι είναι generalists, δηλαδή κατέχουν από όλα τα στοιχεία και μπορούν να λύσουν προβλήματα που κατά τη διάρκεια ανάπτυξης.

Καλλιτέχνες Οι καλλιτέχνες (artists) παράγουν όλο το οπτικοακουστικό κομμάτι του παιχνιδιού, το οποίο είναι βασικό κομμάτι για το χαρακτήρα του παιχνιδιού. Χωρίζονται στις εξής κατηγορίες

- Concept artists οι οποίοι σχεδιάζουν σκίτσα και πίνακες τα οποία παρέχουν στην ομάδα την εικόνα του τελικού παιχνιδιού. Παρέχουν οπτική καθοδήγηση στην ομάδα καθ' όλη τη διάρκεια του κύκλου ανάπτυξης.
- 3D Modelers οι οποίοι είναι υπεύθυνοι για την τρισδιάστατη γεωμετρία του εικονικού κόσμου του παιχνιδιού. Απαρτίζονται από τους foreground modelers οι οποίοι σχεδιάζουν χαρακτήρες, οχήματα, οπλα και αντικείμενα του τρισδιάστατου κόσμου background modelers οι οποίοι σχεδιάζουν την στατικό περιβάλλον πχ κτήρια
- Texture artists οι οποίοι σχεδιάζουν τις δισδιάστατες εικόνες που καλύπτουν τα τρισδιάστατα μοντέλα
- Lighting artists που ορίζουν τις στατικές και δυναμικές πηγές φωτός και δουλεύουν με το χρώμα, την κατεύθυνση και την κατεύθυνση του φωτός.
- Animators οι οποίοι σχεδιάζουν την κίνηση των χαρακτήρων και των αντικειμένων
- Motion capture actors οι οποίοι παρέχουν ακατέργαστα δεδομένα κίνησης για να επεξεργαστούν οι animators και να τα ενσωματώσουν στο παιχνίδι.

- Sound designers οι οποίοι παράγουν τα εφέ και τη μουσική.
- Voice actors τους οποίους η φωνή ηχογραφείται και χρησιμοποιείται για τους χαρακτήρες στο παιχνίδι

Σχεδιαστές Η δουλειά ενός σχεδιαστή παιχνιδιών (game designer) είναι να σχεδιάσει το διαδραστικό τμήμα του παιχνιδιού, το gameplay. Ασχολούνται με τον σχεδιασμό επιπέδων, την ιστορία τις αλληλεπιδράσεις μεταξύ των χαρακτήρων στο παιχνίδι με τους στόχους, σκοπούς και κανόνες του παιχνιδιού. Σχεδιάζουν το κάθε επίπεδο μονδικά και αποφασίζουν για τη γεωμετρία στο περιβάλλον, πότε και που εμφανίζονται χαρακτήρες και διάφορα αντικείμενα, πως γίνονται οι μεταβάσεις μεταξύ διάφορων σκηνών κλπ.

Παραγωγοί Ο ρόλος του παραγωγού (producer) διαφέρει από στούντιο σε στούντιο. Η βασική του δουλειά είναι να προγραμματίζει και να δρομολογεί τις διάφορες εργασίες και να λειτουργεί ως ο συνδετικός κρίκος μεταξύ των ατόμων που παίρνουν ηγετικές αποφάσεις και την ομάδα ανάπτυξης. Οι producers είναι χαρακτηριστικό των AAA εταιριών, όπου υπάρχουν πολλά τμήματα και πολλοί εργαζόμενοι.

Κεφάλαιο 3

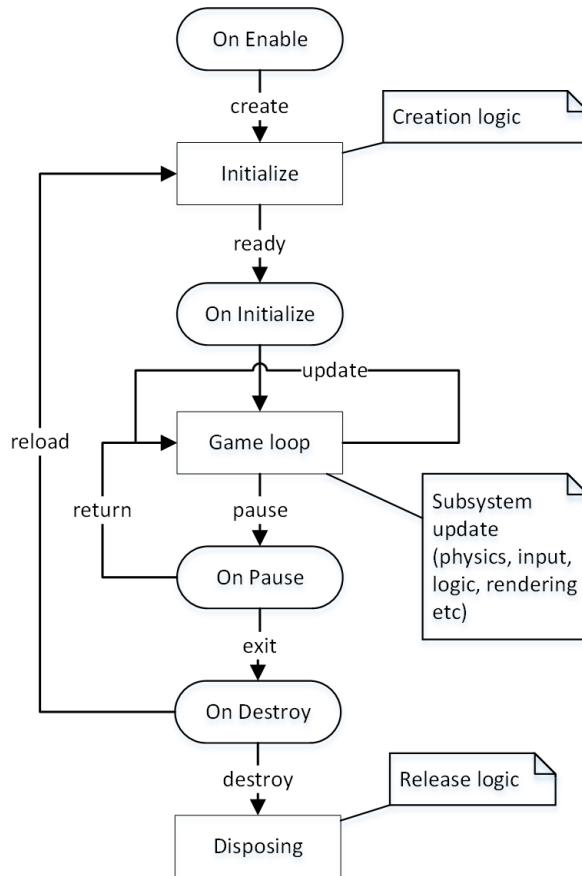
Ο πηρύνας της Μηχανής

Ένα framework το οποίο επεκτείνεται και διακλαδώνεται σε πολλές υπο-βιβλιοθήκες, στηρίζεται στη θεμελιώση ενός framework core, του πηρύνα της βιβλιοθήκη. Το API του πηρύνα πρέπει να είναι εύκολο στην κατανόηση, να αποτελείται από αυτοεπεξηγούμενες αφαιρέσεις, οι οποίες εκθέτουν μόνο τα απολύτος απαραίτητα, ώστε οι υποβιβλιοθήκες που χρησιμοποιούν τον πηρύνα να μην δεσμεύονται σε υλοποιήσεις, να περιορίζει σε συγκεκριμένο pipeline χρήσης για αποφυγή απρόβλεπτων συμπεριφορών και να προσφέρει δυνατότητες επεκτασιμότητας. [8] Ο πηρύνας περιέχει υποσυστήματα τα οποία μπορούν να χρησιμοποιηθούν τη δημιουργία πολλών είδων παιχνιδιών.

3.1 Κύκλος ζωής πηρύνα

Κάθε οντότητα από τη στιγμή της δημιουργίας της στο περιβάλλον της μηχανής μέχρι την καταστροφής της και διαγραφή της από την μνήμη, περνά από κάποια προκαθορισμένα στάδια τα οποία εκτελούνται ανάλογα με την κατάσταση του υλικού και του λογισμικού. Τα στάδια αυτά περιγράφονται στο 3.1

- Initialization εκτελείται μόνο κατά τη δημιουργία
- Game loop εκτελείται συνέχεια σε βρόγχο
- Disposing εκτελείται κατά την καταστροφή



Διάγραμμα 3.1: Κύκλος ζωής του πυρήνα

3.1.1 Βρόγχος παιχνιδιού

Ο βρόγχος του παιχνιδιού (game loop) εγκυάται τη συνεπής ενημέρωση των υποσυστημάτων ανάλογα με το προκαθορισμένο χρονικό βήμα. Χωρίς το ανεξάρτητο σύστημα ενημέρωσης υποσυστημάτων, η ενημέρωση θα γινόταν στον κύκλο εκτέλεσης του επεξεργαστή με αποτέλεσμα την διαφορετική εμπειρία ανά επεξεργαστή και μηχάνημα. Στο game loop ενημερώνονται όλα τα υποσυστήματα 3.2

Το κάθε υποσύστημα έχει διαφορετικές απαιτήσεις για τη βέλτιστη λειτουργία. Το collision detection system μπορεί να χρειάζεται να ενημερώνεται εκατό φορές το δευτερόλεπτο, ενώ το σύστημα τεχνητής νοημοσύνης δύο φορές το δευτερόλεπτο. Η πιο συνηθισμένη τεχνική είναι να υπάρχουν δύο κύρια update loops

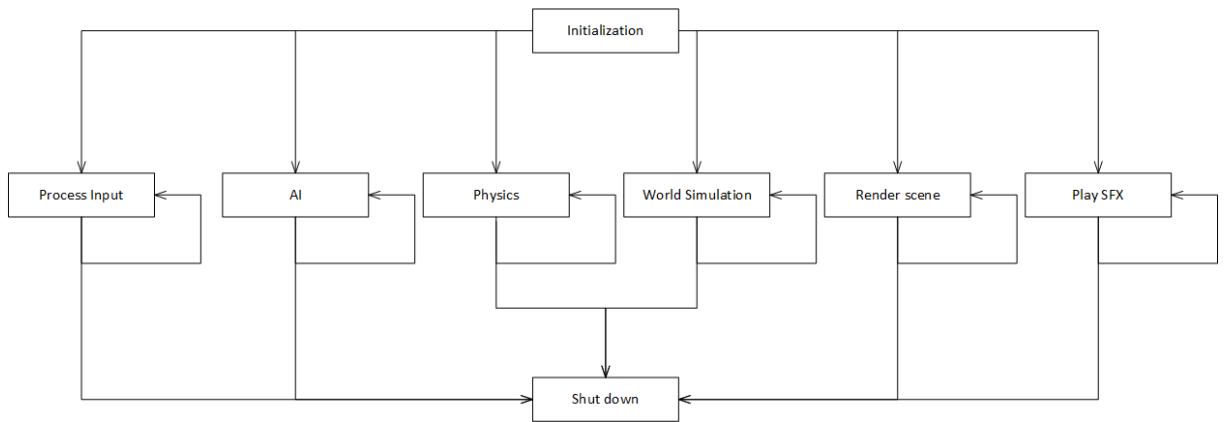
- το game loop στο οποίο ενημερώνονται όλα τα υποσυστήματα, physics, logic, dynamics κλπ
- rendering loop στο οποίο γίνονται οι κλήσεις στην κάρτα γραφικών και τρέχει

στα 50-60 fps

Η διαφορά του χρόνου μεταξύ των ενημερώσεων πρέπει να είναι εγγυημένη για παράδειγμα σε μια μηχανή στην οποία το rendering loop τρέχει στα 60 fps υπάρχει η συχνότητα ενημέρωσης

$$F = 1/T \Rightarrow F = 1000ms/60 \Rightarrow F = 16ms. \quad (3.1)$$

Για να μπορεί να εγγυάται το σύστημα αυτή την αναλογία, πρέπει να μετράει τη διαφορά χρόνου μεταξύ κάθε κλήσης, να εκτελεί τον κώδικα στο συγκεκριμένο loop και για τον υπόλοιπο χρόνο το thread να κοιμάται.



Διάγραμμα 3.2: Game loop

Χρόνος Στα συστήματα πραγματικού χρόνου, η έννοια της διάρκειας και του χρόνου πρέπει να χειρίζεται απομονωμένος. Ο χρόνος του παιχνιδιού είναι ανεξάρτητος από τον πραγματικό χρόνο. Η ενημερώση και η λογική των συστημάτων γίνεται με βάση τη διαφορά χρόνου μεταξύ ενημερώσεων. Με αυτή την στρατηγική, η εμπειρία δεν διαφοροποιείται με λιγότερα fps και ένα animation το οποίο γίνεται render σε πραγματικό χρόνο, μπορεί να παίζει αντίστροφα ή με διπλάσια ταχύτητα, αν το χρονοδιάγραμμα στο οποίο ανταποκρίνεται, χειρίζεται τον χρόνο διαφορετικά. Όλα τα συστήματα ενημερώνονται γραμμικά συναρτήσει μιας αφαίρεσης η οποία περιλαμβάνει τη διαφορά χρόνου.

3.1.2 Υποσυστήματα

Το κάθε υποσύστημα δημιουργείται ενημερώνεται και καταστρέφεται μέσα στο γενικό κύκλο ζωής του πηγύνα. Το κάθε υποσύστημα έχει εσωτερικά το δικό του κύκλο

ζωής και κύκλο ενημέρωσης ο οποίος λειτουργεί μέσα στην έκταση του εξωτερικού κύκλου ζωής.

3.1.3 Τεχνικές Ενημέρωσης Υποσυστημάτων

Τα υποσυστήματα με κάποιο τρόπο πρέπει να ενημερώνονται και να επικοινωνούν μεταξύ τους ή να στέλνουν μηνύματα όταν συμβεί κάποιο γεγονός. Υπάρχουν διάφορές τεχνικές ενημέρωσης υποσυστημάτων.

- Message Pumps: τα υποσυστήματα στέλνουν μηνύματα σε ένα message bus και τα συστήματα ενημερώνονται όταν εξυπηρετείται το μήνυμα. Παραμένουν άεργα εφόσον δεν υπάρχει μήνυμα προς εξυπηρέτηση.
- Call-back driven: τα υποσυστήματα παρέχουν call-back λειτουργικότητα. Δηλαδή τη δυνατότητα να θέσεις τι κώδικας θα εκτελεστεί κατά κάποιο συμβάν. Ένα συμβάν μπορεί να είναι η σύγκρουση μεταξύ δύο αντικειμένων. Ο χρήστης μπορεί να πει ότι όταν συμβεί σύγκρουση μεταξύ του παίχτη και του εχθρού, ο παίχτης θα χάσει ζωή.

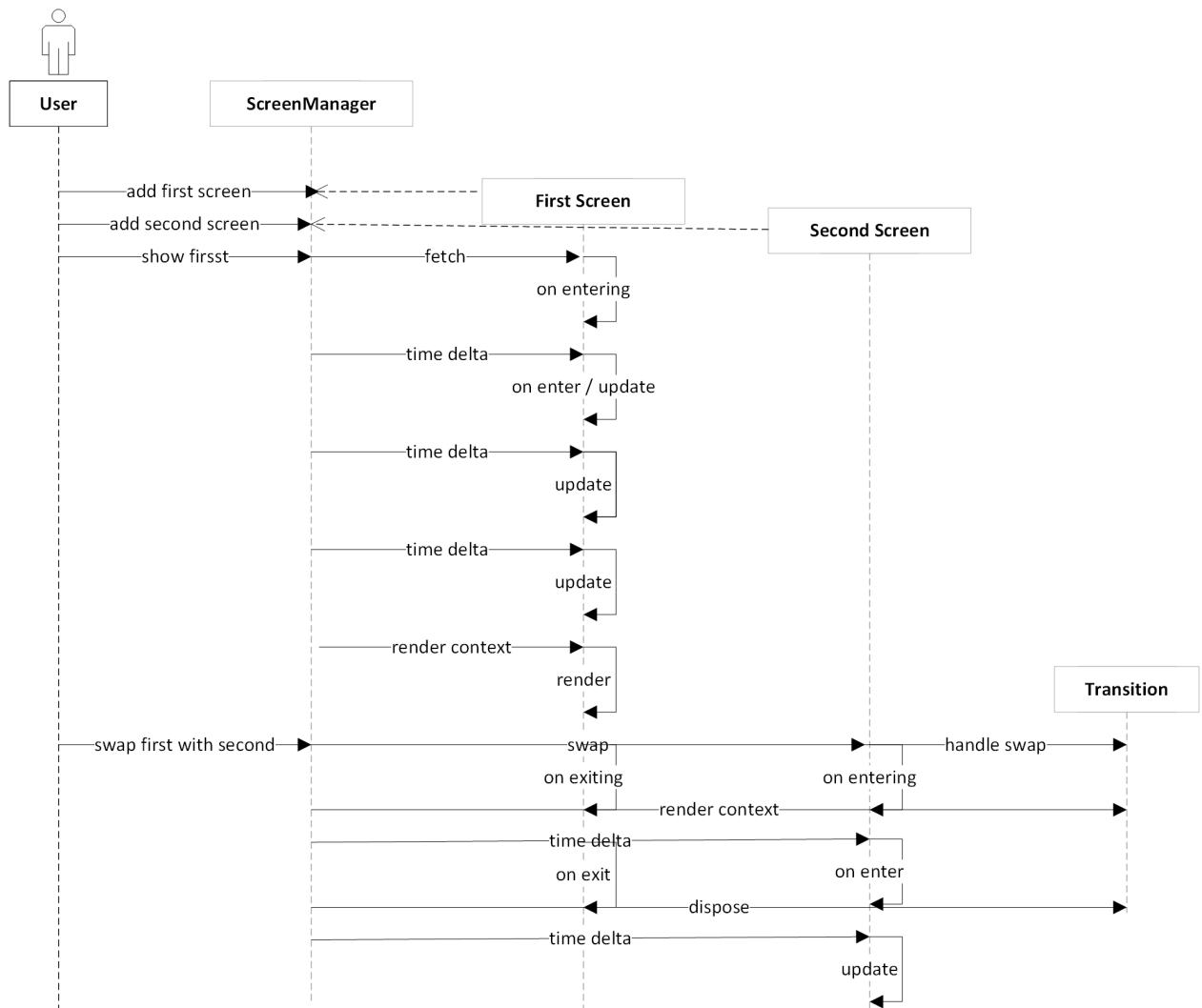
3.2 Διαχείριση οθονών

Σε ένα τυπικό χρόνο εκτέλεσης ενός παιχνιδιού, το παιχνίδι περνά από διάφορες καταστάσεις. Οι καταστάσεις αυτές μπορεί να είναι η προσωρινή παύση, ένα μενού ρυθμίσεων ή ένα gui επιλογών το οποίο επικαλύπτει το τρέχον παιχνίδι ή αποδίδεται στον ίδιο render buffer. Το παιχνίδι χωρίζεται σε σκηνές, σε επίπεδα και σε χάρτες οι οποίες έχουν ξεχωριστό τρόπο απόδοσης στην οθόνη. Μία σκηνή μπορεί να αποδίδεται από διάφορες υποσκηνές οι οποίες δίνουν την ψευδαίσθηση του βάθους στο φόντο. Η σειρά απόδοσης των σκηνών πρέπει να είναι ρυθμιζόμενη και ελεγχόμενη. Η κάθε σκηνή μπορεί να παρομοιαστεί ως μια οθόνη.

3.2.1 Απαιτήσεις

Ο χρήστης του συστήματος έχει πλήρη έλεγχο της κάθε οθόνης-σκηνής και προσαρμόζει τη λογική και την απόδοση ανάλογα. Το κεντρικό σύστημα διαχείρισης προσφέρει τη δυνατότητα προετοιμασίας οθονών, αναζήτησης μέσω κλειδιών, αυτόματη

διαχείριση και εναλλαγή και εξατομίκευση τους. Για τη διατήρηση της συνοχής κατά την εναλλαγή οθόνων, προσφέρεται η δυνατότητα εξατομίκευσης της απόδοσης κατά τις μεταβάσεις. Μια τυπική χρήση του συστήματος παρουσίαζεται στ διάγραμμα ακολουθίας 3.3.



Διάγραμμα 3.3: screensystem sequence

3.2.2 Συστατικά του συστήματος

Κεντρικό σύστημα διαχείρισης Οι λειτουργίες του συστήματος.

- Απόδοση και ενημέρωσης 1-N σκηνών με δυναμικά εναλλασσόμενη σειρά στο πλαίσιο του κύκλου ζωής του πηρύνα.
- Προετοιμασία των οθονών και δυνατότητα εύρεσης χρησιμοποιώντας κλειδιά.

- Εύκολη προβολή, απόκριψη, εναλλαγή οθονών χρησιμοποιώντας κλειδιά.
- Δυνατότητα εξατομίκευσης της απόδοσης κατά τις διάφορες μεταβάσεις των σκηνών.

Screen host

- Παραμετροποίηση συμβάντων κατά τις αλλαγές κατάστασης
- Προσαρμοσμένη λογική και απόδοση
- Εντολές αλλαγής κατάστασης
- Εξατομίκευση απόδοσης κατά την εναλλαγή κατάστασης

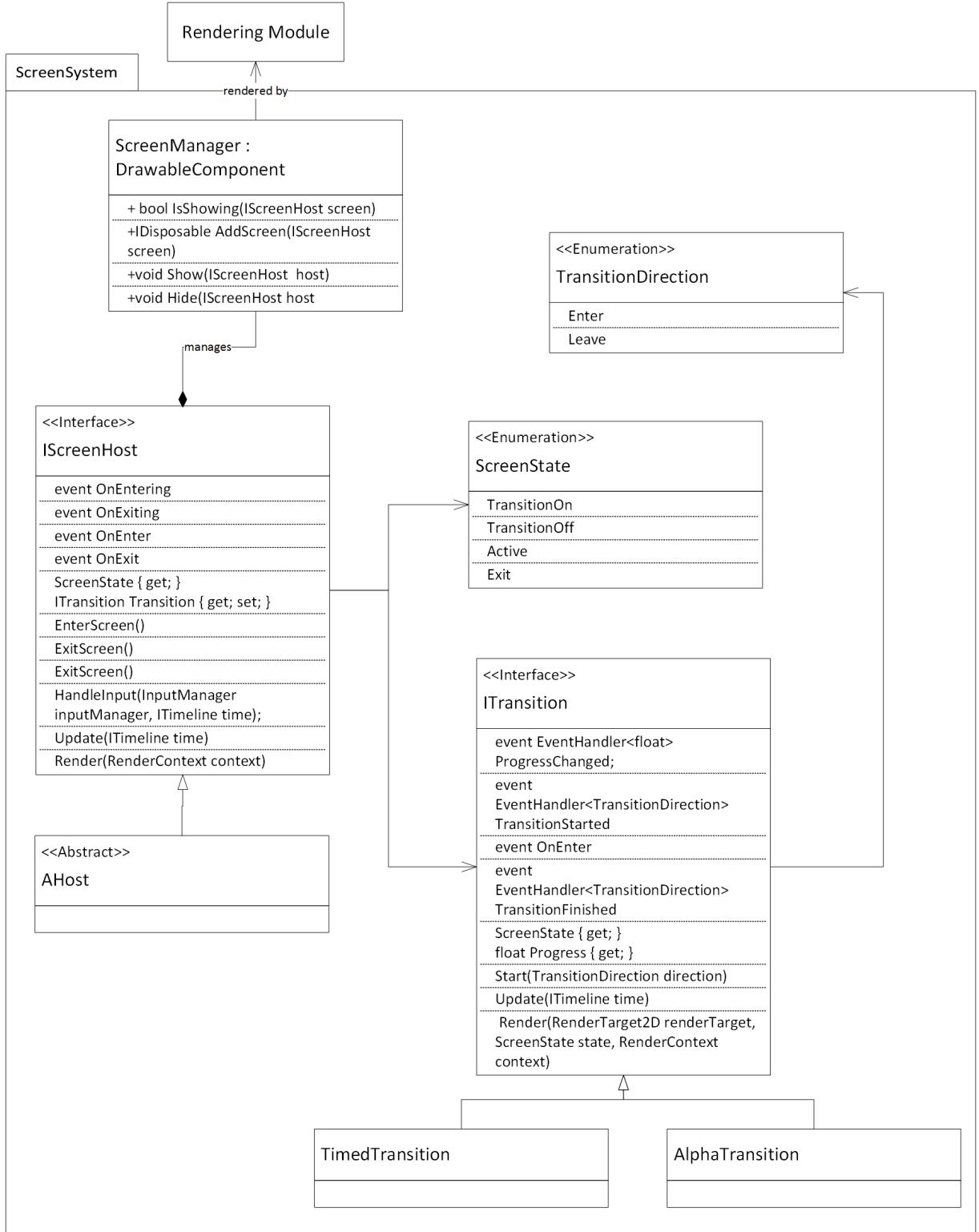
Εναλλαγή οθονών Η εξατομίκευση και παραμετροποίηση της εναλλαγής οθονών λαμβάνει μέρος μέσα στο γενικό πλαίσιο απόδοσης οθονών. Ο εξατομικευτής αναλαμβανει την απόδοση του screen buffer στον οποίο αποδόθηκε η οθόνη, την κατάσταση στην οποία βρίσκεται η οθόνη, την κατεύθυνση και το ποσοστό εξέλιξης της εναλλαγής.

Παράδειγμα κώδικα 3.1: Transition Delegate

```
1 delegate void TransitionRenderAction(ScreenState state, float
progress, RenderTarget2D renderTarget, SpriteBatch batch);
```

3.2.3 Αρχιτεκτονική

Η αρχιτεκτονική του υποσυστήματος παρουσιάζεται στο UML διάγραμμα 3.4.



Διάγραμμα 3.4: Screen System UML

3.2.4 Παράδειγμα χρήσης API

Παράδειγμα κώδικα 3.2: Transition Delegate

```

1 screenHost
2 .AddScreen(
3     "FirstScreen",
4     () => new MyFirstScreen());
5
6 screenHost["FirstScreen"]
7 .Show()
8 .Transition(
9     Transition.WithTime(
10         TimeSpan.FromSeconds(0.5),
11         (state, progress, target, context) =>
12             context.Batch.Draw(target,
13                 (float)Math.Pow(progress - 1.0f, 2) * context.
14                     ScreenWidth,
15                     Color.White * progress
16             )
17         );
18 screenHost["FirstScreen"]
19 .Hide();

```

3.3 Σύστημα εισόδων

3.3.1 Human Interface devices

Η μηχανή πρέπει να είναι σε θέση να διαβάζει, να επεξεργάζεται και να χρησιμοποιεί συσκευές ανθρώπινης διεπαφής. Η διαδικασία ανάγνωσης χωρίζεται στις παρακάτω κατηγορίες:

- Polling: η κατάσταση κάποιον συσκευών (κυρίως της παλιάς σχολής) διαβάζεται ρωτώντας τη συσκευή για την κατάστασή της περιοδικά. Αυτό καταλήγει σε

πλεονασμό γιατί ρωτάει πολλές φορές χωρίς να πάρνει απάντηση και με μικρή καθυστέρηση γιατί η αλλαγή κατάστασης μπορεί να γίνει μεταξύ ερωτήσεων.

- Interrupts (διακοπές): Οι συσκευές στέλνουν δεδομένα μόνο όταν αλλάζει η κατάσταση με κάποιο τρόπο. Ο χρήστης μπορεί γράψει κώδικα και να τον εγγράψει με τρόπο ώστε να εκτελείται μόνο όταν συμβεί κάποια αλλαγή κατάστασης.

3.3.2 Τύποι εισόδου

- Digital Buttons: τα ψηφιακά κουμπιά έχουν δύο καταστάσεις: pressed / not pressed
- Analog axes and buttons: τα αναλογικά επιστρέφουν εύρος τιμών: το βαθμό της πίεσης της σκανδάλης ή τη θέση του μοχλού στο δισδιάστατο άξονα.
- Relative Axes: οι αναφορικοί άξονες επιστρέφουν τιμές σε σχέση με το τελευταίο σημείο στο οποίο έγινε κάποια αλλαγή πχ το ποντίκι επιστρέφει τη διαφορά θέσης σε σχέση με το τελευταίο σημείο στο οποίο μετακινήθηκε.
- Accelerators: Ανιχνεύουν τρισδιάστατες επιταχύνσεις.
- Sensor bars: Αισθητήρες όπως οι κάμερες.
- Touch / gestures: Σε οθόνες αφής όπως στις οθόνες στα έξυπνα τηλέφωνα.

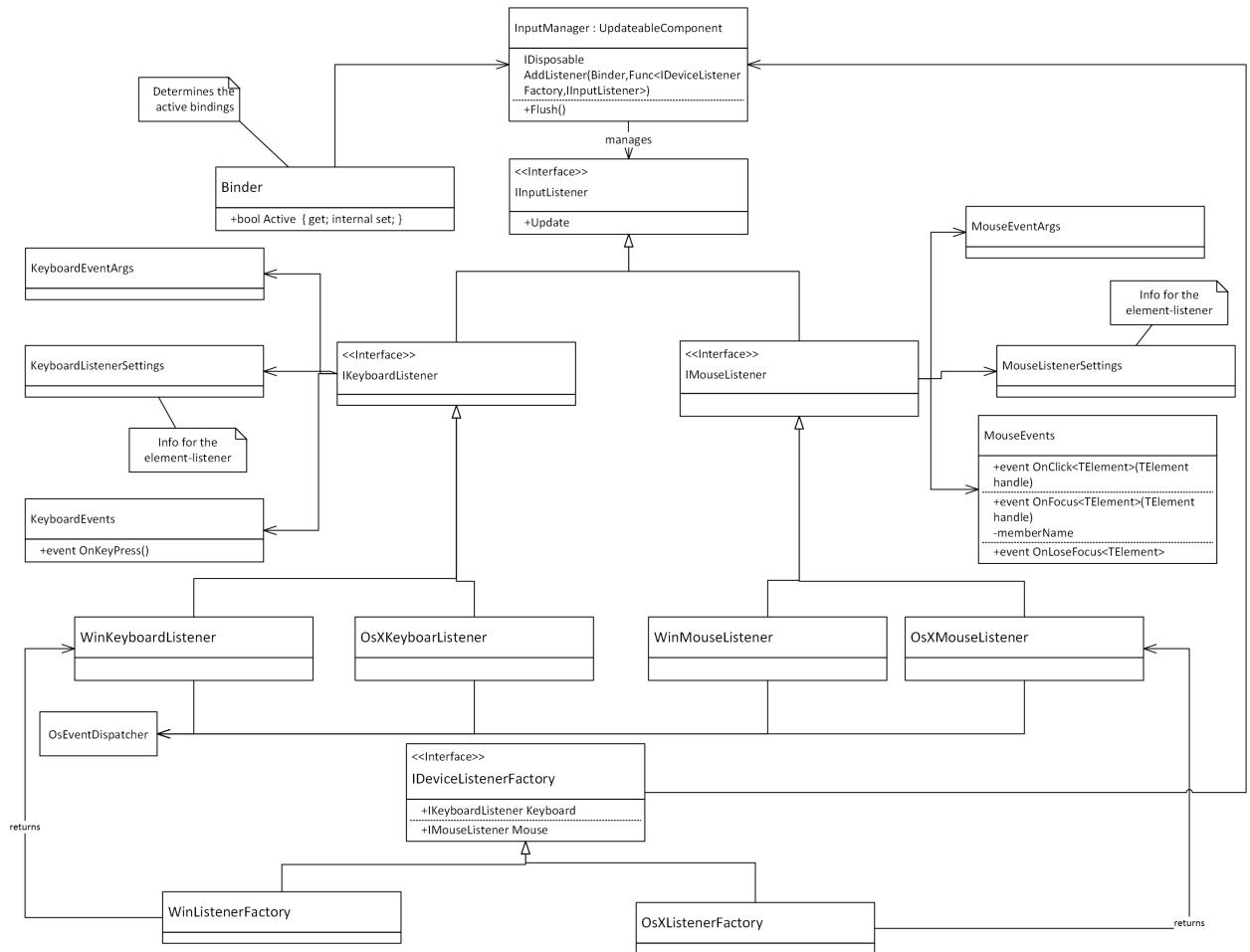
3.3.3 Απαιτήσεις υποσυστήματος

Η οντότητα θέλει να εκτελέσει κώδικα προσανατολισμένο κατά συμβάν εισόδου από συσκευή ανθρώπινης διεπαφής. Ο κώδικας αυτός εκτελείται αντίστοιχα για keyboard-mouse και για gamepad. Ο χρήστης μπορεί να αλλάξει συσκευή διεπαφής ενώ βρίσκεται σε τρέχον παιχνίδι. Η βιβλιοθήκη δεν πρέπει να στηρίζεται σε κανένα υλικό ή συσκευή. Επίσης θέλει τη δυνατότητα να ακυρώνει συμβάντα.

3.3.4 Observers-listeners

Το κάθε υποσύστημα διαχείρισης συσκευών ανθρώπινων διεπαφών αποτελείται από τον διαχειριστή για παράδειγμα τον MouseListener ή Keyboard listener, το οποίο χρησιμοποιεί τις βιβλιοθήκες του τρέχον λειτουργικού για την επιστολή συμβάντων

των διεπαφών. Ανάλογα με την πλατφόρμα στην οποία έγινε compile δημιουργούνται οι αντίστοιχοι listeners μέσω του abstract factory. Ο χρήστης μπορεί να προσκολήσει κάθικα ο οποίος να εκτελείται ανά συμβαν π.χ. στη μετακίνηση του ποντικιού. Οι listeners κατά την εισαγωγή τους, επιστρέφουν ένα αντικείμενο το οποίο μπορεί να χρησιμοποιηθεί για την κατάργησή τους. Οι listeners ανάλογα με τις συνθήκες επιστολής, όπως το δέλτα του χρόνου επιστολής συμβάντος διπλού click είναι 200ms, αποστέλλουν τα συμβάντα στους καταχωρημένους εκτελεστές. Το κεντρικό σύστημα διαχείρισης εισόδου ενημερώνει όλες τις συνδεδεμένες συσκευές εισόδου. Ο εκτελέσιμος κώδικας μπορεί να γραφτεί μια φορά, ανεξαρτήτως συμβάντος, και να προσκοληθεί στο initialization του lifecycle σε υποσυστήματα διεπαφών. Επίσης με τη χρήση των active binders, μπορεί να γίνει απενεργοποίηση των listeners π.χ. όταν είναι ανοιχτό το menu επιλογών, οι ο ActiveBinder του παιχνιδιού είναι απενεργοποιημένος, με αποτέλεσμα τα συμβάτα τα οποία είναι προσκολλημένα στον InputManager να μην εκτελούνται. Στο uml διάγραμμα 3.5 παρουσιάζεται η αρχιτεκτονική του συστήματος των listeners.



Διάγραμμα 3.5: Input System UML

API Στο παράδειγμα 3.3.4 παρουσιάζεται η δημιουργία και καταχώρηση των listeners.

Παράδειγμα κώδικα 3.3: Παράδειγμα καταχώρησης listeners

```

1 inputManager.AddListener(
2 ingameBinder,
3 factory => factory.GamepadListener
4 {
5     Settings = MouseSettings
6         .DoubleClickDelta(Timespan.FromSeconds
7             (0.2))
8         .DragDelta(Timespan.FromSeconds(0.2)),
9     Events = MouseEvents
10        .OnLeftClick(sender, args) => this.Shoot()

```

```

10          . OnLeftDoubleClick( sender , args ) => this . Roll()
11      )
12  });
13 inputManager . AddListener(
14 ingameBinder ,
15 factory => factory . GamepadListener
16 {
17     Settings = GamepadSettings . Default ,
18     Events    = GamepadEvents
19             . OnXButtonPress( sender , args ) => this .
20                     Shoot()
21             . OnYButtonPress( sender , args ) => this .
22                     Roll()
23 });

```

3.4 Φυσική και συγκρούσεις

Η φυσική στα παιχνίδια έχουν ως βάση τα μαθηματικά. Η ανάλυση του συστήματος της φυσικής επικεντρώνεται σε μοντελοποίηση υψηλού επιπέδου. Η υλοποίησή τους βασίζεται υλοποίηση μαθηματικών λειτουργιών γεωμετρίας, γραμμικής άλγεβρας και κινηματικής.

3.4.1 Τα παιχνίδια ως soft real-time simulations

Οι επιστήμονες αποκαλούν τα παιχνίδια soft real-time iterative agent-based computer simulations. Στα περισσότερα παιχνίδια ένα υποσύστημα του πραγματικού κόσμου μοντελοποιείται μαθηματικά, ώστε να μπορεί να αναπαραχθεί και να χειριστεί από τον υπολογιστή. Ένα agent-based simulation είναι μια προσομοίωση η οποία περιγράφει πως αλληλεπιδρούν τα διάφορα αντικείμενα και χαρακτήρες μέσα στον κόσμο. Όλα τα αλληλεπιδραστηκά παιχνίδια είναι temporal simulations δηλαδή ο το μοντέλο του εικονικού κόσμου είναι δυναμικό, αλλάζει με την πάροδο του χρόνου, με βάση τα διάφορα συμβάντα και την εξέλιξη της ιστορίας. Όλα simulations και η επικοινωνία του παιχνι-

διού με τον χρήστη γίνεται σε πραγματικό χρόνο (interactive real-time simulations).

Στον πυρήνα όλων των συστημάτων πραγματικού χρόνου υπάρχει το at least 24 fps deadline δηλαδή για να δημιουργείται η ψευδαίσθηση της κίνησης, η οθόνη θα πρέπει να ανανεώνεται τουλάχιστον 24 φορές το δευτερόλεπτο. Φυσικά υπάρχουν και άλλα ειδή deadlines. Για να θεωρείται η προσομοίωση φυσικής σταθερή, πρέπει να ενημερώνεται τουλάχιστον 120 φορές το δευτερόλεπτο, οι audio buffers 60 φορές το δευτερόλεπτο για να αποτρέπονται δυσλειτουργίες του συστήματος διαχείρησης ήχου.

Ένα soft real time system είναι ένα σύστημα στο οποίο χαμένες ενημερώσεις δεν είναι καταστροφικές. Τα μαθηματικά μοντέλα τα οποία απαρτίζουν το σύστημα μπορεί να είναι είτε αριθμητικά είτε αναλυτικά. Τα αριθμητικά μοντέλα μπορούν να αξιολογηθούν για κάθε τιμή της ανεξάρτητης μεταβλητής ενώ οι τιμές του αναλυτικού μοντέλου καθορίζονται διακρίτα κατά τη διάρκεια της προσομείωσης και είναι πιο συχνά γιατί η επόμενη κατάσταση της προσομείωσης καθορίζεται από την εισαγωγή δεδομένων σε πραγματικό χρόνο από το χρήστη. [1]

3.4.2 Βασικές έννοιες φυσικής του συστήματος

Οι οντότητες της φύσικης έχουν τις παρακάτω ιδιότητες για τη προσομοίωσή τους στο υποσύστημα φυσικής.

- Mass: Η μάζα στη φυσική συνδέεται με δύο έννοιες, την αδράνεια της μεταφορικής κίνησης και τη βαρύτητα. Η μάζα είναι μια ορισμένη ποσότητα η οποία χρησιμοποιείται για την περιγραφή ενός συστήματος.
- Density: Η πυκνότητα εκφράζει τη μάζα του υλικου που περιέχεται σε μία μονάδα όγκου.
- Force: Σε ότι αφορά τα ελεύθερα σώματα, η δύναμη είναι γενικά η αιτία μεταβολής της κινητικής τους κατάστασης, δηλαδή αυτή που τα επιταχύνει ή τα επιβραδύνει. Αυτό ισχύει και για την περιστροφή τους, που μπορεί να επιταχυνθεί ή να επιβραδυνθεί. Για σώματα που δεν είναι ελεύθερα να κινηθούν με όλους τους τρόπους, αυτά δηλαδή που είτε είναι αναρτημένα κάπου και μπορούν να κινηθούν μόνο γύρω από σημείο ή άξονα ή σε προκαθορισμένη τροχιά, καθώς και σε όσα εφαρμόζονται δυνάμεις τριβής ή γενικά αντιδράσεις στήριξης.

- Torque: Ροπή δυνάμεως ως προς σημείο είναι το διανυσματικό φυσικό μέγεθος που έχει μέτρο ίσο προς το γινόμενο της δύναμης επί την (κάθετη) απόσταση της δύναμης από το σημείο. Κατά όμοιο τρόπο ροπή δυνάμεως ως προς άξονα είναι το διανυσματικό μέγεθος που έχει ως μέτρο το γινόμενο της δύναμης επί την (κάθετη) απόσταση της δύναμης από τον άξονα, και φορέα τον άξονα.
- Impulse: Η ώθηση είναι φυσικό μέγεθος που ισοδυναμεί με την μεταβολή της ορμής ενός σώματος στο οποίο εφαρμόζεται μία δύναμη για κάποιο χρονικό διάστημα. Ισούται με το γινόμενο της δύναμης που ασκείται στο σώμα επί τον συνολικό χρόνο εφαρμογής της
- Restitution: To law of restitution δηλαδή ανάκτυση με βάση το κέρδος. Η υποχρέωση της οντότητας να αποζημιώσει από τα κέρδη τη για κάποιο γεγονός.
- Damping: Η απόσβεση, η επιρροή εντός η κατά ενός συστήματος ταλάντωσης που έχει ως αποτέλεσμα τη μείωση, τον περιορισμό η την πρόληψη των ταλαντώσεών του. Σε φυσικά συστήματα , απόσβεση παράγεται με διαδικασίες που διαχέουν την ενέργεια που αποθηκεύεται στο ταλάντωση.

3.4.3 Οντότητες του συστήματος

- Shape Ένα αντικείμενο το οποίο αντιπροσωπεύει ένα γεωμετρικό σχήμα όπως κύκλο, πολύγωνο κλπ
- World Η συλλογή των bodies, fixtures και constraints και η λογική της αλληλεπίδρασής τους.
- World Solver Ο solver αναλύει την προσομοίωση και εκτελείται ανεξάρτητα από το χρόνο εκτέλεσης του προγράμματος.
- Fixture To fixture δένει ένα body με ένα shape και προσθέτει επιπλέον ιδιότητες όπως πυκνότητα, τριβή και αποκατάσταση. Το fixture εντάσσει ένα σχήμα στο σύστημα συγκρούσεων ώστε να αλληλεπιδρά με τα υπόλοιπα σχήματα στον κόσμο.
- Body To body περιέχει της πληροφορίες ένος αντικειμένου του οποίου δεν βλέπεις ούτε συγκρούεσαι. Έχουν τις παρακάτω ιδιότητες

- mass Το βάρος
- velocity η ταχύτητα και η κατεύθυνση της κίνησης υπό μορφή διανύσματος.
- rotation inertia πόσος κόπος χρειάζεται για να ξεκίνησει η περιστροφή ή η κίνηση
- angular velocity πόσο γρήγορα και σε ποια κατεύθυνση περιστρέφεται
- που βρίσκεται στο σύστημα καρτεσιανών συντεταγμένων
- angle υπό ποια γωνία βρίσκεται

Οι τύποι των bodies είναι:

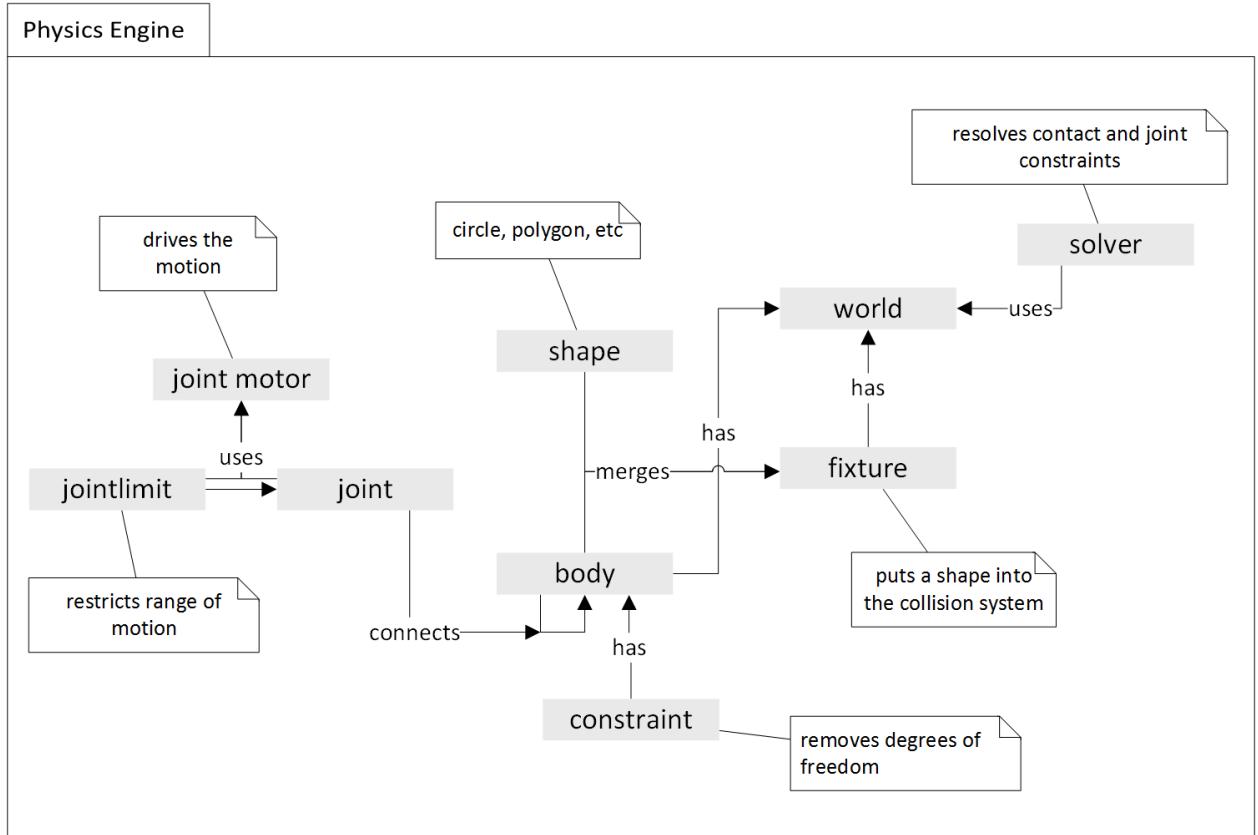
- Static Στατικά στο σύστημα συντεταγμένων. Δεν ανταποκρίνεται σε εξωτερικές δυνάμεις.
- Dynamic Είναι μέρος του συστήματος συγκρούσεων. Ανταποκρίνεται σε εξωτερικές δυνάμεις και κανονικά σε όλες τα μέρη της προσωμοίωσης.
- Kinematic Κινείται ανάλογα με το προκαθορισμένο script ταχύτητας. Δεν ανταποκρίνεται σε εξωτερικές δυνάμεις.
- Constraint Περιορισμός κατά την προσωμοίωση του body.
- Joint To joint συνδέει δύο ή περισσότερα bodies μεταξύ τους.
- Joint Motor Ο οδηγός της κίνησης των συνδεδεμένων με joint bodies.
- Joint Limit Περιορίζει το εύρος κίνησης του joint motor.

3.4.4 Αρχιτεκτονική

Η σχέσης μεταξύ των οντοτήτων παρουσιάζονται στο διάγραμμα 3.6

3.5 Διεπαφή χρήστη

Η επικοινωνία του χρήστη με τη μηχανή γίνεται συνήθως με human interface devices. Η επικοινωνία αυτή αντί να περιορίζεται σε ένα απλό input signal, μπορεί να γίνει εύκολα, αυτονόητα, αποτελεσματικά και φιλικά προς το χρήστη. Το User interface



Διάγραμμα 3.6: Physics System

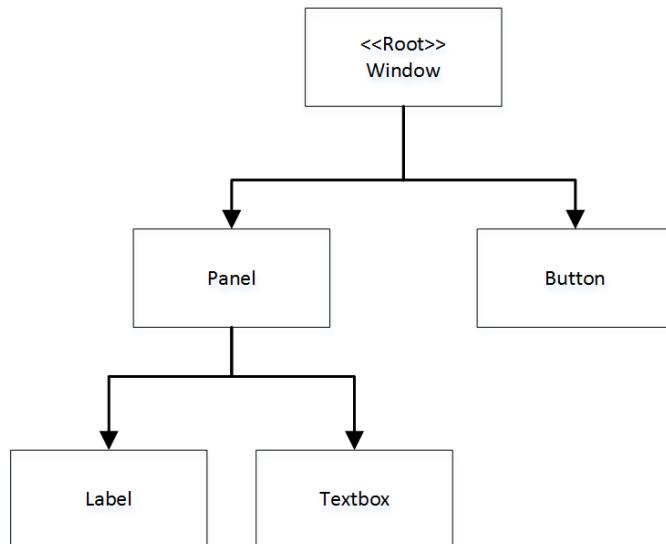
(διεπαφή χρήστη) ονομάζουμε το σύνολο γραφικών στοιχείων, τα οποία εμφανίζονται στην οθόνη κάποιας ψηφιακής συσκευής (π.χ. H/Y) και χρησιμοποιούνται για την αλληλεπίδραση του χρήστη με τη συσκευή αυτή. Παρέχει μέσω γραφικών ενδείξεις και εργαλεία προκειμένου ο χρήστης να φέρει εις πέρας κάποιες επιθυμητές λειτουργίες.

3.5.1 Απαιτήσεις

Κατά το σχεδιασμό ενώς παιχνιδίου συχνά δημιουργείται η ανάγκη για διεπαφή χρήστη. Η δομή του UI, το στυλ της απόδοσης (χρώμα, textures, fonts κλπ), η απόδοσης (rendering) και η συμπεριφορά και ο εκτελέσιμος κώδικας κατά τα συμβάντα της διεπαφής πρέπει να είναι αποσυνδεδεμένα έτσι ώστε να μην επηρεάζει το ένα το άλλο. Επίσης χρησιμεύει όταν το UI προορίζεται για διάφορες αναλόσεις και Dpi. Μια καλή στρατηγική στην κατανόηση και επίλυση προβλημάτων είναι η εύρεση παρόμοιων και σχετικών προβλημάτων. Στο σχεδιασμό ιστοσελίδων η HTML χρησιμοποιείται και τη δομή της σελίδας, το CSS για το στυλ και η Javascript για τη συμπεριφορά.

3.5.2 Δομή στη μνήμη

Όταν ζητηθεί από τον περιηγητή να φορτώσει μια ιστοσελίδα στη μνήμη από κάπιον web server, αναλύει το html αρχείο και οργανώνει τα στοιχεία σε δομή δέντρου B-Tree. Λόγω της οργάνωση σε δέντρο, δημιουργούνται σχέσεις μεταξύ των στοιχείων.



Διάγραμμα 3.7: UI B-Tree

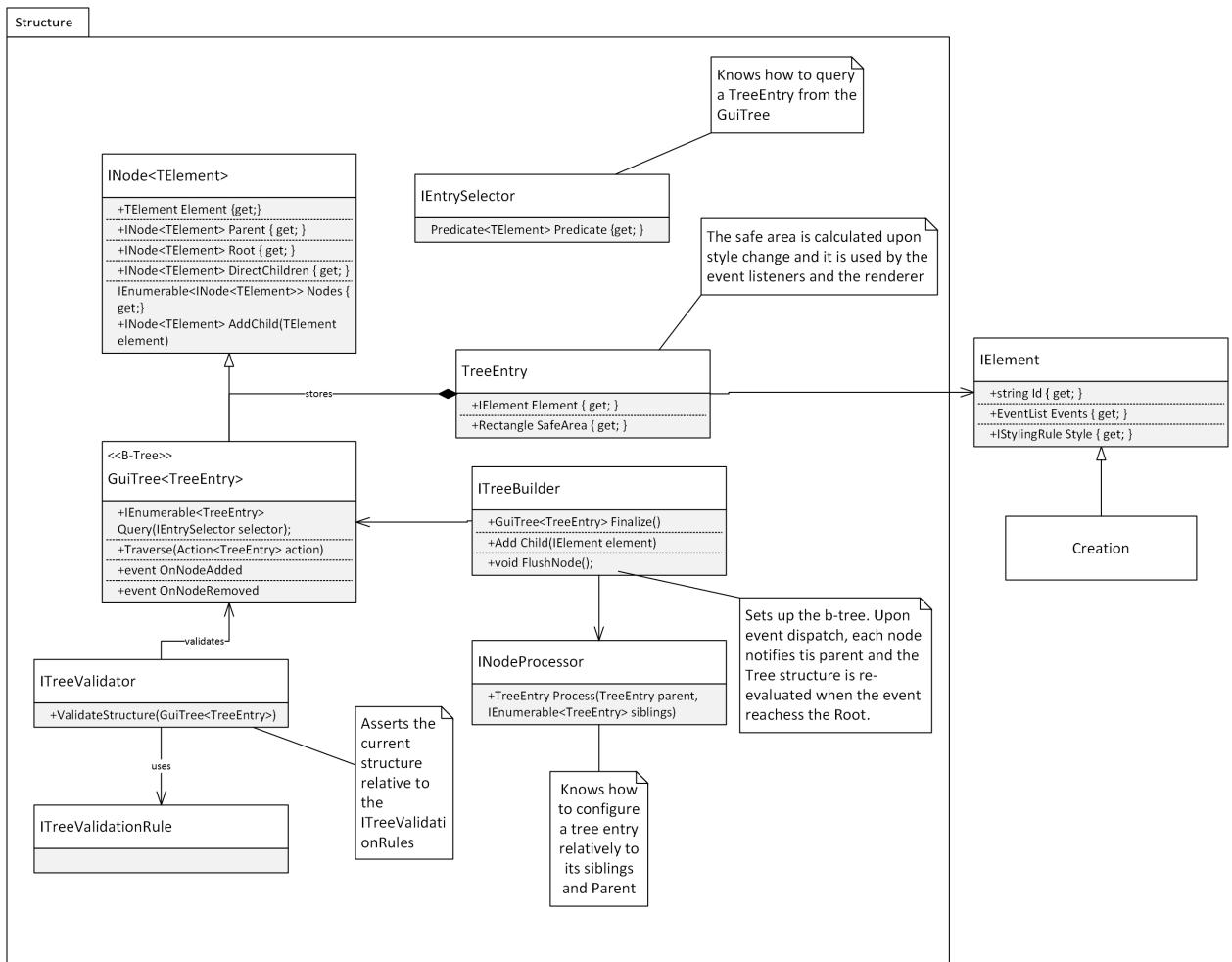
Στο παράδειγμα 3.7 το window είναι το root του δέντρου. Το window έχει δύο παιδιά, ένα panel και ένα button. Το button έχει με και αυτό δύο παιδιά: ένα textbox και ένα label. Τα button, textbox και label είναι leafs του δέντρου, το panel είναι node, και το panel και button siblings.

Γιατί b-tree; Η οργάνωση σε b-tree έχει τα παρακάτω πλεονεκτήματα.

- Ευκολία κατά την απόδοση στην οθόνη. Λόγω της συγκεκριμένης δομής, τα στοιχεία μπορούν να στοιχιθούν και να αποδοθούν ανάλογα με τη σχέση τους με συγγενικά στοιχεία. Καθώς διαβαίνεται το δεντρό, το κάθε στοιχείο αποδίδεται μέσα στο πλαίσιο του πατέρα του και σε πιο πάνω επίπεδο, ούτως ώστε να δημιουργείται η ψευδαίσθηση του βάθους, δηλαδή ότι το στοιχείο παιδί βρίσκεται "πάνω" άπο τον πατέρα.
- Καλύτερη απόδοση στο rendering. Όταν κάποιος κόμβος δεν διατέμνεται με το πλάισιο του παράθυρου, δεν απόδίδεται στην οθόνη και το traversing του δέντρου σταματά.

- Αποσαφήνιση σειράς συμβάντων. Κατά τα συμβάντα στο UI, ένα πλαίσιο μπορεί να καλύπτει περισσότερα από ένα συμβάντα. Ένα στοιχείο έχει τεμνόμενα πλαίσια όταν οποίο βρίσκεται μέσα σε ένα άλλο στοιχείο. Το συμβάν γίνεται στο στοιχείο το οποίο βρίσκεται τελευταίο στην ιεραρχία.
- Προχωρημένη και γρήγορη επιλογή στοιχείων. Η επιλογή των στοιχείων γίνεται με προχωρημένα κριτήρια, όπως τη σχέση ενός στοιχείου με συγγενικά στοιχεία στη δομή, και σε λογαριθμητικό χρόνο.

Στο UML διάγραμμα 3.8 παρουσιάζεται το υποσύστημα το οποίο είναι υπεύθυνο για το κτίσμα του UI στη μνήμη.



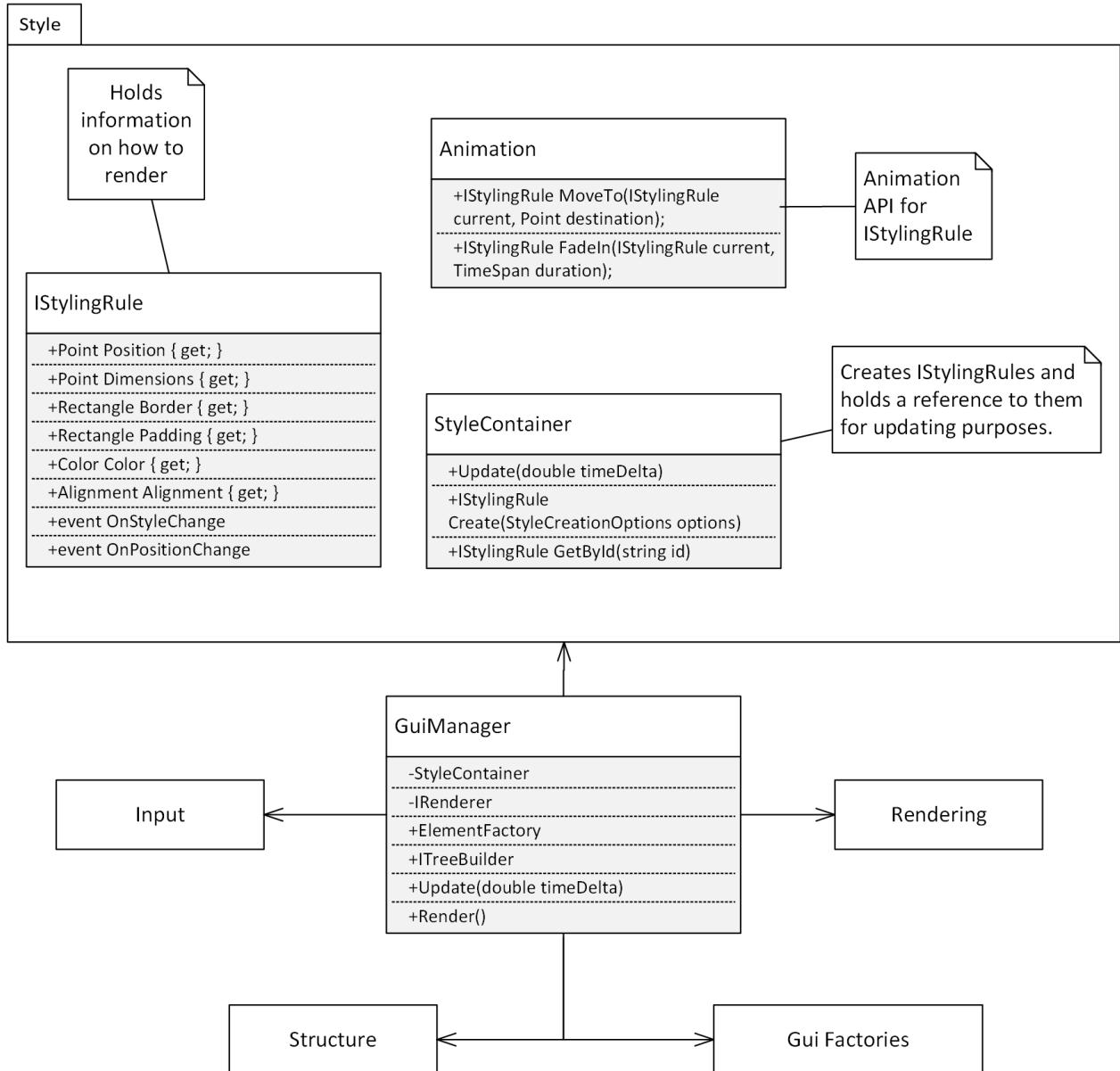
Διάγραμμα 3.8: UI Structure

3.5.3 Το σύστημα

Το σύστημα ανθρώπινης διεπαφής χωρίζεται στα παρακάτω υποσυστήματα

- Οργάνωσης δομής, το οποίο κτίζει και αποθηκεύει την ιεραρχία των στοιχείων στη μνήμη
- Style definition το οποίο επαυξάνει την απόδοση με animations, textures κλπ
- Factory μέσα από το οποίο ο χρήστης δημιουργεί στοιχεία. Στο υποσύστημα αυτό, όπως και το input listener factory είναι χρησιμοποιεί abstract factory για δημιουργεία στοιχείων ανεξαρτήτου πλατφόρμας, και fluent builder για τη δημιουργία στοιχείων με φιλικό στον χρήστη API.
- Επεκτίνει και χρησιμοποιεί με το input system για να συνδέει συμβάντα με στοιχεία
- Επεκτίνει και χρησιμοποιεί το rendering module για να αποδίδει το δεντρο στην οθόνη.

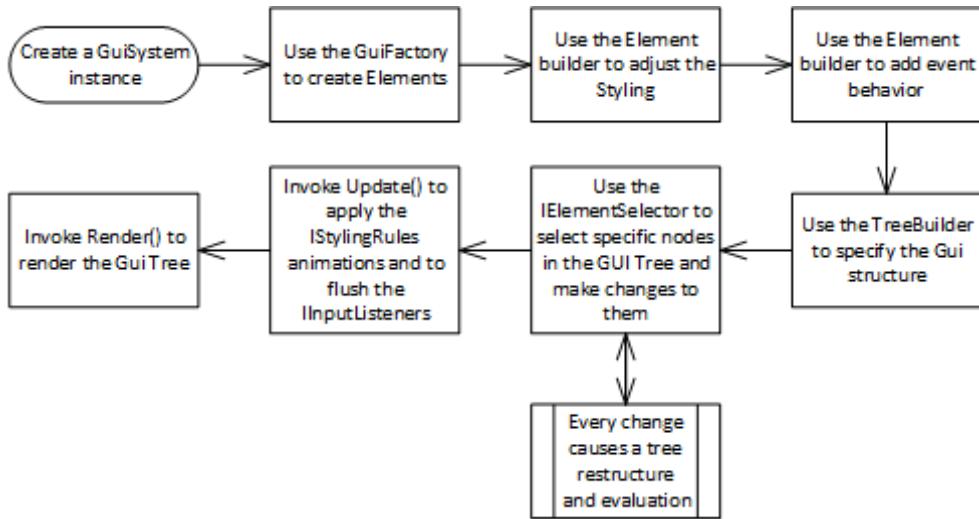
Οι σχέσεις μεταξύ των υποσυστημάτων του συστήματος UI παρουσιάζονται στο UML διάγραμμα 3.9.



Διάγραμμα 3.9: UI System UML

3.5.4 Τρόπος χρήσης

Η σειρά προετοιμασίας διαμόρφωσης και παρουσίασης του UI φαίνεται στο διάγραμμα 3.10.



Διάγραμμα 3.10: UI Usage Diagram

3.6 Διαχείριση πόρων

Τα παιχνίδια απαρτίζονται από πολλα είδη δεδομένων, είτε αυτά είναι τρισδιάστατα μοντέλα, είτε είναι textures είτε αρχεία ήχου κλπ. Η μηχανή πρέπει να είναι σε θέση να φορτώνει στη μνήμη και να αφαιρεί αυτά τα δεδομένα δυναμικά. Χρειάζεται λοιπόν κάποιου είδους asset / resource manager, ο οποίος χειρίζεται το σύστημα αρχείων του λειτουργικού. Ο asset manager πρέπει να προσαρμόζεται ανάλογα με το λειτουργικό στο οποίο τρέχει και να λαμβάνει υπόψη τις ιδιαιτερότητες των διάφορων συστημάτων αρχείων. Πρέπει να υποστηρίζεται το ασύγχρονο φόρτωμα δεδομένων, για να μην μπλοκάρεται η ροή του παιχνιδιού.

Ο διαχειριστής πόρων (resource manager) απαρτίζεται από δύο μέρη. Το πρώτο χειρίζεται εργαλεία τα οποία χειρίζονται assets και τα μεταφράζουν σε μια μορφή η οποία είναι επεξεργάσιμη από τη μηχανή. Το δεύτερο διαχειρίζεται τα assets, φροντίζει να είναι διαθέσιμα όταν χρειάζονται και όταν δεν θα ξαναχρησιμοποιηθούν φροντίζει να αφαιρέσει από τη μνήμη. Τα περισσότερα assets δεν φορτώνονται στη μνήμη με την κανονική τους μορφή. Τα assets περνούν από κάποιο conditioning pipeline για να μετασχηματιστούν σε ένα format το οποίο μπορεί να επεξεργαστεί η μηχανή. Δεν είναι αρκετό απλά ένα μοντέλο, η μηχανή πρέπει να ξέρει για τι προορίζεται. Περιέχουν metadata το οποίο περιγράφει το πως η μηχανή θα χειριστεί το συγκεκριμένο asset. Χρειάζεται μια resource database η οποία ξέρει πως να χειρίζεται διάφορα είδη assets σε μια μορφή την οποία η μηχανή καταλαβαίνει. Μπορεί απλά το κάθε asset να περιέχει

μαζί του κάποιο αρχείο xml το οποίο εξηγεί τι θα κάνει η μηχανή μαζί του. Επίσης είναι χρήσιμο να υποστηρίζει versioning. Η βάση μπορεί να είναι είτε SQL είτε απλά ένα b-tree με buffer blocks και δυνατότητα προσπέλασής του.

Σχεδιασμός Resource Pipeline

- Granular resources: resources τα οποία συνδέονται με τις οντότητες στο παιχνίδι.
- Συνδεση με κώδικα: μπορεί εύκολα να γίνει η σύνδεση των δεδομένων με τον πηγαίο κώδικα
- Ευκόλο export δεδομένων σε διάφορα formats.
- Συμβατότητα με άλλα formats ή δυνατότητα μετατροπής τους σε format το οποίο είναι συμβατό με τη μηχανή.
- Εύκολο build, αφού η μηχανή αναλαμβάνει τις εξαρτήσεις μεταξύ δεδομένων και κώδικα.

3.6.1 Ευθύνες του offline resource manager

- Exporters η δυνατότητα μετατροπής native format σε format το οποίο μπορεί να τροποποιηθεί από τη μηχανή.
- Resource Compilers κατά το export χρειάζεται να γίνει κάποιου είδους καμουφλάρισμα των δεδομένων ώστε να συμβαδίζουν με την μηχανή.
- Resource Linkers πολλές φορές περισσότερα από ένα αρχεία χρειάζονται να συνδεθούν για να δημιουργηθεί ένα χρήσιμο πακέτο. Η είναι υπεύθυνη στο να βρίσκει εξαρτήσεις και να ενώνει κομμάτια ώστε να δημιουργεί πακέτα έτοιμα χρησιμοποιηθούν.

3.6.2 Ευθύνες του Runtime- Resource Manager

- Εξασφαλίζει ότι στη μνήμη υπάρχουν μόνο μοναδικά resources και περισσότερα από ένα του ίδιου τύπου.

- Διαχειρίζεται το πόσο χρόνο είναι διαθέσημα στη μνήμη Φορτώνει και ξεφορτώνει από τη μνήμη
- Χειρίζεται composite resources δηλαδή resource το οποίο αποτελείται από περισσότερες από μία. Ένα τρισδιάστατο μοντέλο για παράδειγμα, αποτελείται από ένα mesh, materials, textures, skeletal animations κλπ
- Διαχειρίζεται την ακεραιότητα των αναφορών, δηλαδή λαμβάνει υπόψη τις υπεξαρτίσεις και τις αλληλοεξαρτίσεις και φροντίζει να μην υπάρχουν προβλήματα.
- Διαχειριση μνήμης.
- Επιστρέπει την τροποποίηση των δεδομένων αφού φορτωθούν στη μνήμη
- Προσφέρει τη δυνατότητα ασύγχρονης φόρτωσης για παραλληλισμό ενεργειών.

Οργάνωση αρχείων και καταλόγων Συνήθως γίνεται δενδροειδής οργάνωση για τα resources. Πολλές φορές για σκοπούς επίδοσης, πολλά αρχεία είναι συμπιεσμένα σε ένα για να γινέται πιο γρήγορη προσπέλαση. Μια μηχανή μπορεί να χρησιμοποιήσει ήδη υπάρχων formats ή κάποιο προσαρμοσμένο για τις ανάγκες της μηχανής.

Τεχνικές διαχείρισης resources στη μνήμη Μια συνηθισμένη τεχνική είναι η οργάνωση σε δομή δεδομένων hashtable με κλειδιά και τιμές, όπου κλειδί είναι το μοναδικό χαρακτηριστικό, ένα GUID ή η διαδρομή του αρχείου στο δίσκο. (flyweight pattern). Όταν το παιχνίδι χρειάζεται κάποιο resource, η μηχανή ελέγχει αν υπάρχει το κλειδί. Αν υπάρχει τό επιστρέφει και αν όχι τότε φορτωνεί στο hashtable.

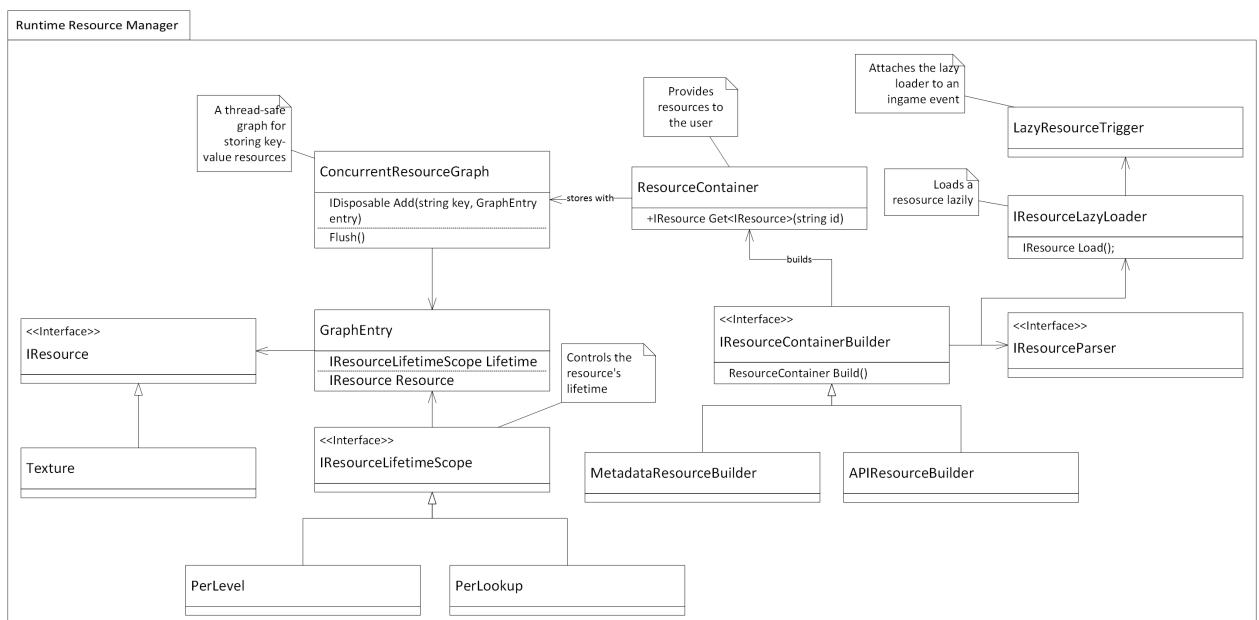
Διαχείριση του κύκλου ζωής στη μνήμη

- Global resources: κάποια resources φορτώνονται στην αρχή του παιχνιδιού και χρειάζονται συνέχεια.
- Level resources: άλλα χρειάζονται στο scope συγκεκριμένου level-layout
- Short-living resources: κάποια χρειάζονται για ένα συγκεκριμένο σκοπό κατα τι διάρκεια π.χ. μια σκηνή η οποία εμφανίζεται μια φορά κατα τη διάρκεια ενός level.

- Live streamed resources: αρχεία μουσική και ηχητικών εφέ τα οποία διαβάζονται από το δίσκο δυναμικά και φορτώνονται σε chunks.

Τεχνικές διαχείρισης κύκλου ζωής Υπάρχουν διάφορες τεχνικές για τη διαχείρηση του κύκλου ζωής των resources. Μια καλή τεχνική είναι να φυλάονται τα references στα αντικείμενα κάθε φορά που γίνεται προσπέλαση στη μνήμη π.χ. στην αρχή κάθε level. Μετράται πόσες φορές ζητήθηκε συγκεριμένο resource. Βήμα 1: Βρίσκουμε όλα τα resources που χρειάζονται για τη συγκεκριμένη σκηνή και αυξάνουμε τον μετρητή τους κατά 1. Αφαιρούμε στα υπόλοιπα 1. Βήμα 2: Σε όσα ο μετρητής έγινε 1, τα φορτώνουμε στη μνήμη και σε όσα έγινε 0 τα αφαιρούμε από τη μνήμη.

Αρχιτεκτονική του Runtime Resource Manager Η αρχιτεκτονική του runtime resource manager παρουσιάζεται στο UML διάγραμμα 3.11



Διάγραμμα 3.11: Runtime Resource UML

3.7 Τεχνητή νοημοσύνη

Η τεχνητή νοημοσύνη χρησιμοποιείται για να παράγει ευφυείς συμπεριφορές κυρίως χαρακτήρες εκτός του παίχτη (NPCS). Οι αλγόριθμοι της τεχνικής νοημοσύνης προσπαθούν να προσομοιώσουν την ανθρώπινη συμπεριφορά και βασίζονται σε τεχνικές από θεωρία ελέγχου, ρομποτική και γενικά της πληροφορικής. Στα ηλεκτρονικά

παιχνίδια η νοημοσύνη συμβάλει στο πιο ρεαλιστικό gameplay μέσα στους περιορισμούς του περιβάλλοντος. Η προσέγγιση είναι εντελός διαφορετική από την τεχνητή νοημοσύνη σε άλλα πεδία, γιατί οι ικανότητες του υπολογιστή πρέπει να είναι ήπιες ώστε να δώσει σε ανθρώπινους παίκτες μια αίσθηση δικαιοσύνης και ικανοποίησης.

3.7.1 Δέντρα συμπεριφορών

Το δέντρο συμπεριφορών (behavior tree) είναι ένα δέντρο με ιεραρχικά συνδεδεμένους κόμβους για έλεγχο της ροής της διαδικασίας λήψης αποφάσεων μιας οντότητας με νοημοσύνη. Στα φύλλα του δέντρου βρίσκονται οι εντολές και η συμπεριφορά της οντότητας. Τα κλαδιά τα δημιουργούν διάφοροι τύποι κόμβων οι οποίοι περιέχουν εντολές και περιορισμούς για τη διάβαση του δέντρου.

Η βασική οντότητα του δέντρου είναι ο κόμβος. Ο κάθε κόμβος αξιολογείται σε κάθε βήμα διάβασης του δέντρου και επιστρέφει την κατάστασή του:

- Success: Ο κόμβος αξιολογήθηκε με επιτυχία
- Failure: Ο κόμβος αξιολογήθηκε με αποτυχία
- Running: Ακόμη να τελειώσει η αξιολόγηση του κόμβου.

Η διάβαση του δέντρου προχωράει ανάλογα με την κατάσταση του προηγούμενου κόμβου. Οι κόμβοι αξιολόγησης λειτουργούν σαν λογικές πύλες.

- Sequence: Προσομοιώνει την πύλη AND, για να αξιολογηθεί ο κόμβος κλαδί με επιτυχία, πρέπει όλοι οι κόμβοι του κλαδιού να αξιολογηθούν επιτυχώς.
- Selector: Προσομοιώνει την πύλη OR. Η αξιολόγηση του κλαδιού θεωρείται επιτυχημένη στην πρώτη αξιολόγηση ενός δέντρου με επιτυχία.
- Random Selector: Ίδια συμπεριφορά με το selector, απλά η αξιολόγηση των κόμβων γίνεται με τυχαία σειρά.
- Random Sequence: Ίδια συμπεριφορά με το sequence, απλά η αξιολόγηση των κόμβων γίνεται με τυχαία σειρά.

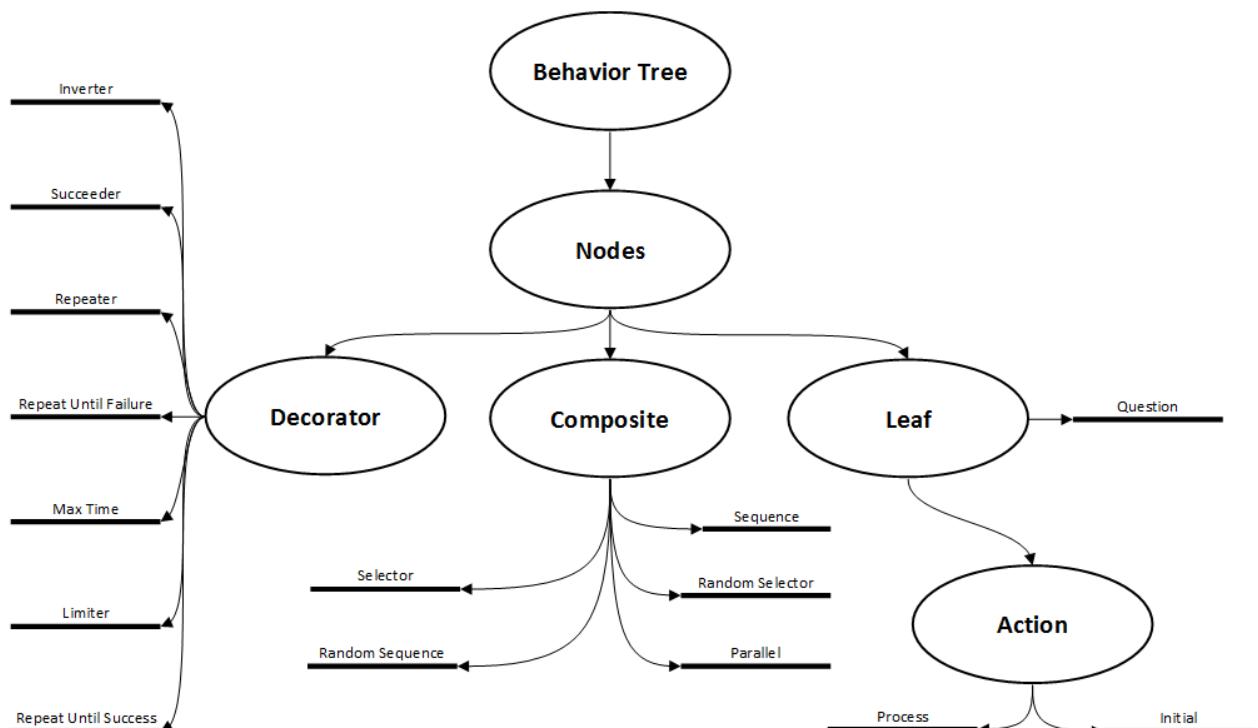
Η συμπεριφορά και αξιολόγηση των κόμβων μπορεί να μεταβληθεί χρησιμοποιώντας το decorator pattern.

- Inverter: Ο κόμβος ο οποίος ειναι decorated από τον inverter, επιστρέφει το αντίστροφο αποτέλεσμα κατά την αξιολόγηση.
- Repeat Until Success: Ο κόμβος αξιολογείται ως Running εφόσων αξιολογηθεί με αποτυχία.
- Repeater: Η αξιολόγηση του κόμβου επαναλαμβάνεται ανάλογα με το προκαθορισμένο αριθμό επαναλήψεων
- Max Time: Η αξιολόγηση γίνεται μέσα σε στα πλαίσια χρονικού διαστήματος.

Τα φύλλα του δέντρου καθορίζουν την πραγματική συμπεριφορά της νοημοσύνης.
Τα φύλλα μπορούν να είναι:

- Ερωτήσεις. Υπάρχει το συγκεκριμένο αντικείμενο στο χάρτη;
- Συμπεριφορά. Προχώρα μπροστά.

Στο mind map 3.12 παρουσιάζονται οι σχέσεις και οι τύποι των κόμβων.



Διάγραμμα 3.12: Behavior Trees

API Το API του gem engine παρέχει fluent tree builder για εύκολη δημιουργία δέντρων στη μνήμη με κώδικα και tree visualizer για απόδοση του δέντρου στην οθόνη με αξιολόγηση και ενημέρωση πραγματικού χρόνου για αποσφαλμάτωση. Στο παρακάτω παράδειγμα θα γινει μοντελοποίηση της συμπεριφοράς ενώς χαρακτήρα ο οποίος προσπαθεί να βρει την έξοδο σε ένα λαβύρινθο με εχθρούς.

Παράδειγμα κώδικα 3.4: Behavior Tree Builder

```

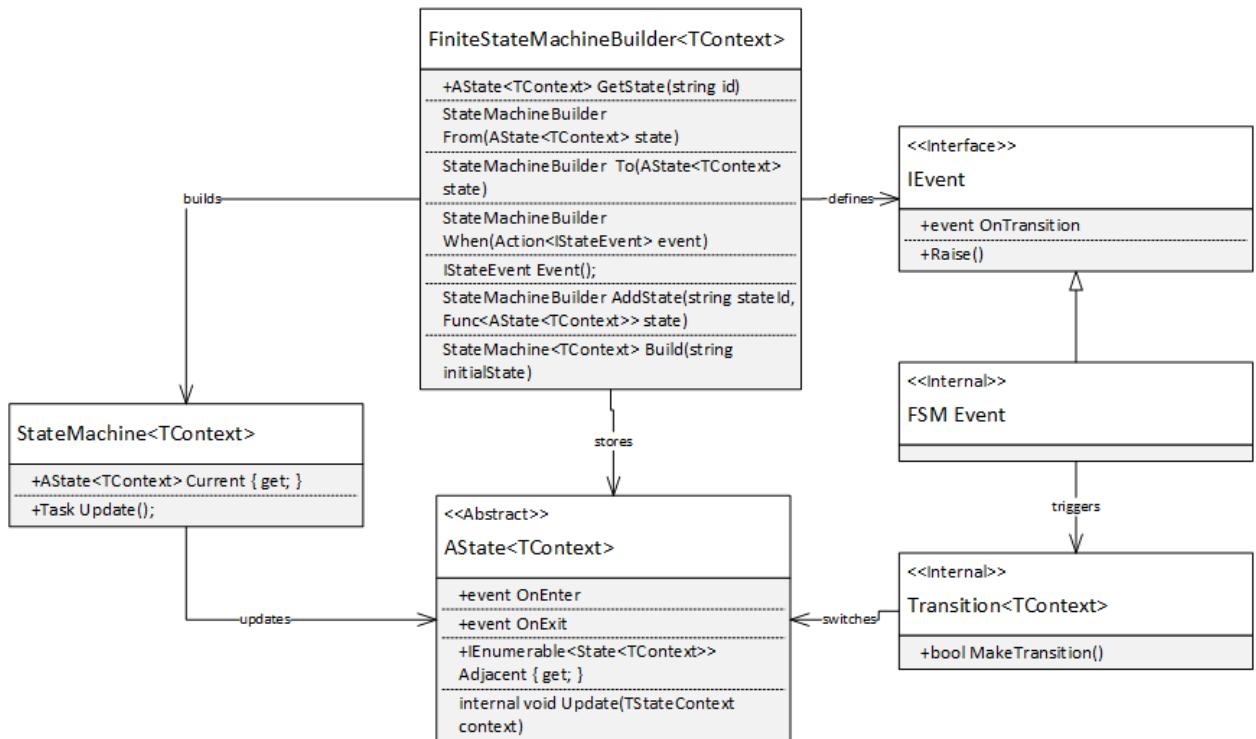
1 behaviorBuilder
2   . Selector(name: "Move\u2014to\u2014end")
3     . Decorate(For.RepeatUntilFailure)
4       . Sequence(name: "try\u2014walk")
5         . Question(CheckNextTile, name: "next\u2014tile\u2014empty?")
6         . Behavior(MoveOneTile, name: "go\u20141\u2014step\u2014forward")
7       . Sequence("key\u2014obstacle")
8         . Question(CheckTileForKey, name: "found\u2014key?")
9         . Selector("handle\u2014key")
10      . End
11      . Sequence
12        . DecorateFor(Decorator.AlwaysSuccess)
13          . Behavior(RunAway)
14        . End
15      . End
16    . Tree;

```

3.7.2 Finite State Machines

Finite state machine είναι μια αφηρημένη μηχανή η οποία μπορεί να βρίσκεται σε μία από ένα πεπερασμένο αριθμό καταστάσεων και με την επέλευση κάποιου γεγονότος μπορεί να αλλάξει από μια κατάσταση σε άλλη. Κατάσταση (state) είναι η περιγραφή της κατάστασης στην οποία βρίσκεται το σύστημα. Γεγονός (event) δράση η οποία οδηγεί σε αλλαγή κατάστασης. Transition (μετάβαση) είναι ένα σύνολο δρά-

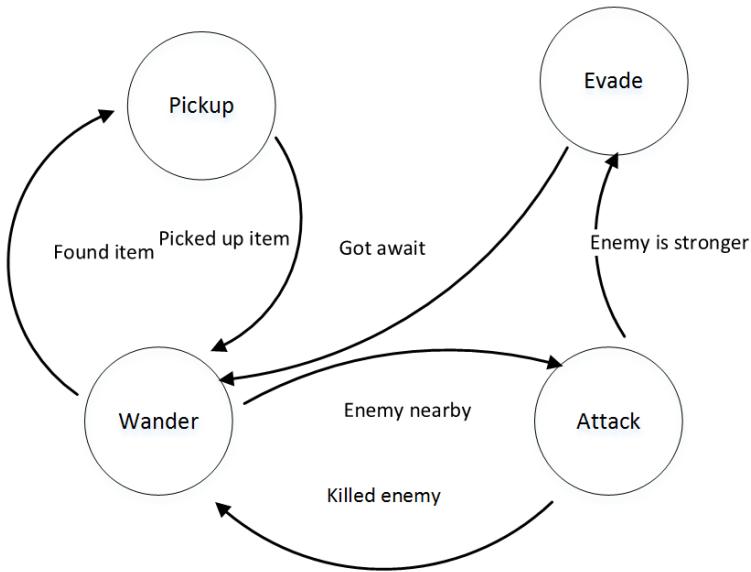
σεων που πρέπει να εκτελεστούν όταν πληρείται μια προϋπόθεση ή όταν συμβεί ένα γεγονός. Στο UML 3.13 παρουσιάζονται οι σχέσεις μεταξύ των οντοτήτων του finite state machine.



Διάγραμμα 3.13: State Machine UML

Στο παράδειγμα 3.14 μοντελοποιούνται οι καταστάσεις και τα γεγονότα τα οποία οδηγούν σε αλλαγή κατάστασης ενός χαρακτήρα με νοημοσύνη. Ο χαρακτήρας αυτός περιπλανιέται τυχαία, όταν συναντήσει αντικείμενο το μαζεύει και όταν συναντήσει εχθρό τον πολεμά. Αν ο εχθρός είναι πιο δυνατός, τότε αποφεύγει την μάχη φεύγοντας.

Η σύνδεση του finite state machine γίνεται μέσω του builder. Στον builder γίνεται καταχώρηση των καταστάσεων μέσω μιας μεθόδου-factory και δημιουργία αυτοενεργοποιημένων συμβάντων ή συμβάντων τα οποία ενεργοποιούνται από τον χρήστη τα οποία οδηγούν σε αλλαγή κατάστασης. Μετά γίνεται το χτίσημα του FSM και η ενημέρωση του με βάση το γενικότερο πλαίσιο.



Διάγραμμα 3.14: State Machine Example

Παράδειγμα κώδικα 3.5: State machine

```

1 fsmBuilder
2 . AddState("PickUp", () => new PickupState())
3 . AddState("Wander", () => new WanderState())
4 //with lifetime
5 . AddState("Attack", () => ioc.Resolve<AttackState>())
6 . AddState("Evade", () => new EvadeState())
7
8 var attackEvent = fsmBuilder
9 . From("Wander")
10 . To("Pickup")
11 . When(context=> context.FoundItem) //auto triggered
12
13 var attackEvent = fsmBuilder
14 . From("Wander")
15 . To("Attack")
16 . Event() //manually triggered
17
18 world.OnCollision += (sender, args)=>
19 {
20     if(args.CollidedType is Enemy)
  
```

```
21     attackEvent.Trigger();  
22 }  
23  
24 // emitted  
25 var playerBehaviorFsm = fsmBuilder.Build("Active");  
26 await playerBehaviorFsm.Update(playerContext);
```

Κεφάλαιο 4

Δικτύωση

Δικτύωση στα ηλεκτρονικά παιχνίδια έχουμε όταν περισσότεροι από ένας παίχτες σε διαφορετικές πλατφόρμες ή υπολογιστές, μοιράζονται και αλληλεπιδρούν στο ίδιο εικονικό περιβάλλον.

4.1 Το πρόβλημα

4.1.1 Περιγραφή του προβλήματος

Διάφοροι παίχτες σε διάφορα σημεία του πλανήτη θέλουν να μοιραστούν ένα εικονικό περιβάλλον σε πραγματικό χρόνο με σκοπό την συνεργασία ή την αντιπαλότητα. Ο κόσμος είναι ένα υπερσύνολο του offline κόσμου με επιπλέων στοιχεία κοινωνικοποίησης όπως η επικοινωνία μέσω μηνυμάτων ή φωνής.

4.1.2 Κατανόηση του προβλήματος

Ένας εικονικός κόσμος, περιλαμβάνει πολλές οντότητες οι οποίες αλληλεπιδρούν μεταξύ τους μέσω των μηχανισμών, νόμων και κανόνων που διέπουν τον κόσμο. Παράδειγμα μηχανισμού είναι η προσομοίωση του φυσικού κόσμου, όπου οι οντότητες αναποκρίνονται σε νόμους της φυσικής.

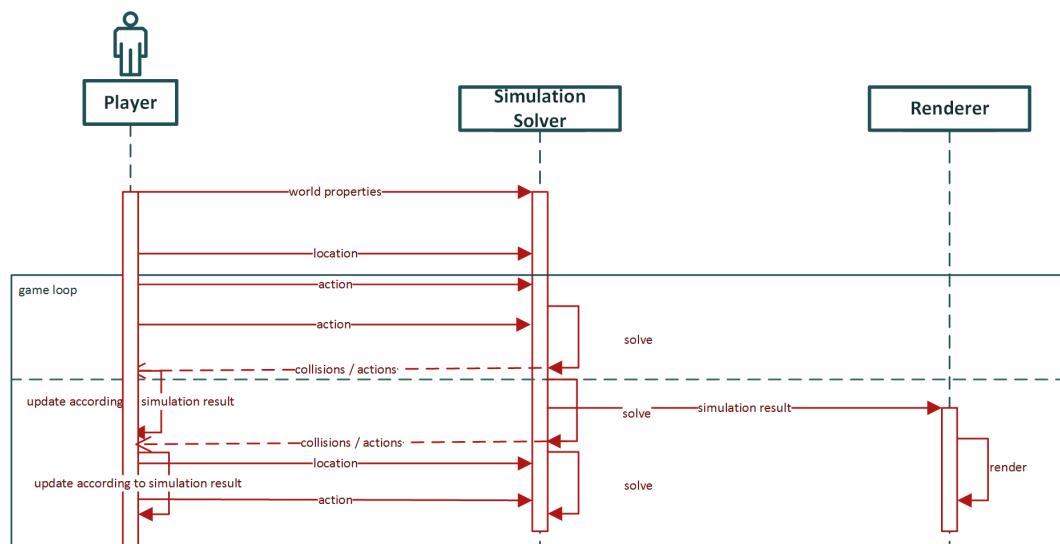
Κατά την ενημέρωση του κόσμου, ο προσομοιωτής χρησιμοποιώντας τους νόμους, τους κανόνες και τους μηχανισμούς που διέπουν τον κόσμο, παίρνει ως είσοδο τις οντότητες, τα ειδικά βάρη των ιδιοτήτων τους, την απόλυτη θέση τους στο σύστημα συντεταγμένων του κόσμου, και το χρονικό διάστημα της προσομοίωσης και αναλύει την

προσομοίωση. Η ανάλυση της προσομοίωσης για να είναι επιτρεπτά ακριβής πρέπει να γίνεται περίπου 80-100 φορές το δευτερόλεπτο. [αναφορά σε πηγή]

Στο τέλος της προσομοίωσης, η μηχανή γραφικών αποτυπώνει τον κόσμο στις εξόδους με αναφορές σε στατικά assets, σε αλγόριθμους παραγωγής δυναμικών assets για την αναπαράσταση του κόσμου.

4.1.3 Εξαγωγή απαιτήσεων

Με βάση το πρόβλημα καταλήγουμε στο παρακάτω διάγραμμα ακολουθίας 4.1.



Διάγραμμα 4.1: Network Sequence Diagram

Βλέποντας το διάγραμμα καταλαβαίνουμε ότι σε η διαδικασία rendering περιλαμβάνει στατικά στοιχεία και αλγόριθμους, τα οποίοι μπορούν να φορτώνονται τοπικά στον κάθε υπολογιστή, και δεν είναι απαραίτητα για την επίλυση της προσομοίωσης. Τα απαραίτητα στοιχεία για την προσομοίωση τα οποία πρέπει να μοιράζονται μεταξύ των παιχτών είναι:

- Οι ιδιότητες της οντότητας με βάση τους νόμους του κόσμου. Οι ιδιότητες αυτές δεν ενημερώνονται συχνά.
 - Οι αλλαγές στο σύστημα συντεταγμένων και οι διάφορες ενέργειες της κατευθυνόμενης οντότητας κατά την πάροδο του χρόνου. Οι αλλαγές τοποθεσίας και ενέργειες γίνονται πολλές φορές ανά δευτερόλεπτο. Η προσομοίωση για να είναι ακριβής πρέπει να ενημερώνεται για τις διάφορες ενέργειες σε πραγματικό

χρόνο.

4.2 Εκπόνηση σχεδίου

Η ανάγκη αποστολής πολλών μηνυμάτων ανά δευτερόλεπτο οδηγεί στη χρήση των network sockets. Τα sockets χρησιμοποιούνται ως ένα IP και Port και επιτρέπουν την αποστολή και παραλαβή μηνυμάτων με βάση κάποιου πρωτόκολλου.

4.2.1 Επιλογή πρωτοκόλλου

- TCP (transmission control protocol), είναι το πιο συχνά χρησιμοποιημένο πρωτόκολλο. Η διασύνδεση με TCP είναι αξιόπιστη και τα μηνύματα παραλαμβάνονται στη σειρά αποστολής. Η αξιοπιστία όμως έρχεται με ένα μικρό κόστος απόδοσης.
- UDP To UDP (user diagram protocol) δεν περιλαμβάνει την αξιοπιστία και την εγγύηση της αλληλουχίας μηνυμάτων. Η απουσία λειτουργιών όμως, το κάνει το πιο γρήγορο σε αποστολή μηνυμάτων πρωτόκολλο.

Η επιλογή πρωτοκόλλου γίνεται ανάλογα με το γενικότερο πλαίσιο και τη συγκεκριμένη χρήση του πακέτου αποστολής. Στην αποστολή της τοποθεσίας, το οποίο γίνεται 20φορές / δευτερόλεπτο, η αξιοπιστία δεν είναι το βασικότερο, αλλά η απόδοση. Στην αποστολή των στοιχειών του χρήστη κατά την έναρξη, ή ενώς γραπτού μηνύματος πρέπει να είναι αξιόπιστη.

4.2.2 Επιλογή αρχιτεκτονικής δικτύου

Οι αρχιτεκτονικές δικτύου χωρίζονται ανάλογα με το που γίνεται η επίλυση και επεξεργασία της προσομοίωσης.

- Client-server model: στο οποίο ο client απλά κάνει render και το μεγαλύτερο κομμάτι της λογικής και της προσωμοίωσης τρέχει στον server. Ο server στέλνει οδηγίες στον client για το τι να κάνει render και ο client απλά υπακούει.
- Client on top of server model: ο client είναι και server, δηλαδή οι μηχανές που έχουν τον client έχουν και τον server.

- Peer-to-peer: οι μηχανές συμπεριφέρονται μερικώς ως clients και μερικώς ως servers, δηλαδή έχουν και στοιχεία λογικής και επεξεργασίας.

Η client-server αρχιτεκτονική εφαρμόζεται σε παιχνίδια τα οποία περιλαμβάνουν πολύ μεγάλους κόσμους και η προσομοίωση περιλαμβάνει αλληλεπιδράσεις μεταξύ πολλών οντοτήτων. Οι προσωμοιώσεις αυτές γίνονται σε εξειδικευμένες μηχανές και όχι στον προσωπικό υπολογιστή του κάθε χρήστη γιατί χρειάζονται μεγάλους υπολογιστικούς πόρους. Επίσης ο server μπορεί να λύσει race conditions και να λειτουργήσει ως "διαιτητής" μεταξύ δύο οντοτήτων οι οποίες ζητούν πρόσβαση στο ίδιο σύστημα κατά το ίδιο χρονικό διάστημα. Οι αρχιτεκτονικές στις οποίες ο client περιλαμβάνει στοιχεία επεξεργασίας του κοινόχρηστου κόσμου, είναι πιο δύσκολες στην ανάπτυξη συντήρηση γιατί δημιουργούνται race conditions στο χρονικό διάστημα αποστολής-παραλαβής μηνυμάτων που προορίζονται σε αλληλοεξαρτώμενες λειτουργίες.

4.3 Σχεδίαση του framework

Κατά το σχεδιασμό διαδικτυακών παιχνιδιών πολλά μοτίβα και στερεότυπα κώδικα επαναλαμβάνονται χωρίς να συμβάλλουν στην λογική ή μηχανισμούς. Σκοπός του framework είναι να μειώσει και να απλοποιήσει τον επαναλαμβανόμενο κώδικα και να προμηθεύσει τον χρήστη με μοτίβα και μοντέλα ούτως ώστε να εστιάσει μόνο στα απαραίτητα.

4.3.1 Έννοιες

Serialization Σε μια αντικειμενοστραφή γλώσσα χρησιμοποιούνται κλάσεις για να μοντελοποιήσουν το πρόβλημα. Όμως τα αντικείμενα τον κλάσεων δεν μπορούν να σταλούν μέσω network socket στην αρχική τους μορφή, πρέπει να γίνει μια μετατροπή σε binary για να είναι συνεπής με το format των δεδομένων που αποστέλλονται μέσω του socket. Κατά την αποστολή έχουμε το serialization των αντικείμενων, και κατά την παραλαβή το deserialization του πακέτου στο αντικείμενο που αντιπροσωπεύει.

4.3.2 Τύποι μηνυμάτων

Οι διάφοροι τύποι μηνυμάτων χρησιμοποιούνται για να ξεχωρίσουν την κατάσταση στην οποία βρίσκεται ο client ή o server.

- Αλλαγή κατάστασης
 - Connecting
 - Connected
 - Disconnecting
 - Disconnected
- Data
- ErrorMessage
- WarningMessage
- VerboseMessage
- ConnectionApproval
- DiscoveryRequest
- DiscoveryResponse

Οι τύποι μηνυμάτων μπορούν να μοντελοποιηθούν ως ένα enum και να προστεθεί η πληροφορία τους στο πρώτο byte του serialized μηνύματος. Το πρώτο byte του παραληφθέντα μηνύματος περιλαμβάνει τον τύπο του μηνύματος.

4.3.3 NetworkManager

Ο network manager είναι υπεύθυνος για την διαχείριση των sockets, την αποστολή / παραλαβή μηνυμάτων, ενθυλακώνει λειτουργίες για την διαδικτύωση και είναι ο πηρύνας του framework και επεκτίνεται από τον client, server οι οποίοι προσθέτουν λειτουργίες.

4.4 Υλοποίηση

4.4.1 Τρόπος χρήσης

Η διαδικασία χρήσης 4.2 πρέπει να είναι εξορθολογιστική και ξεκάθαρη για εύ-κολη και γρήγορη ανάπτυξη και περιοριστική για αποφυγή bugs και προβλημάτων.

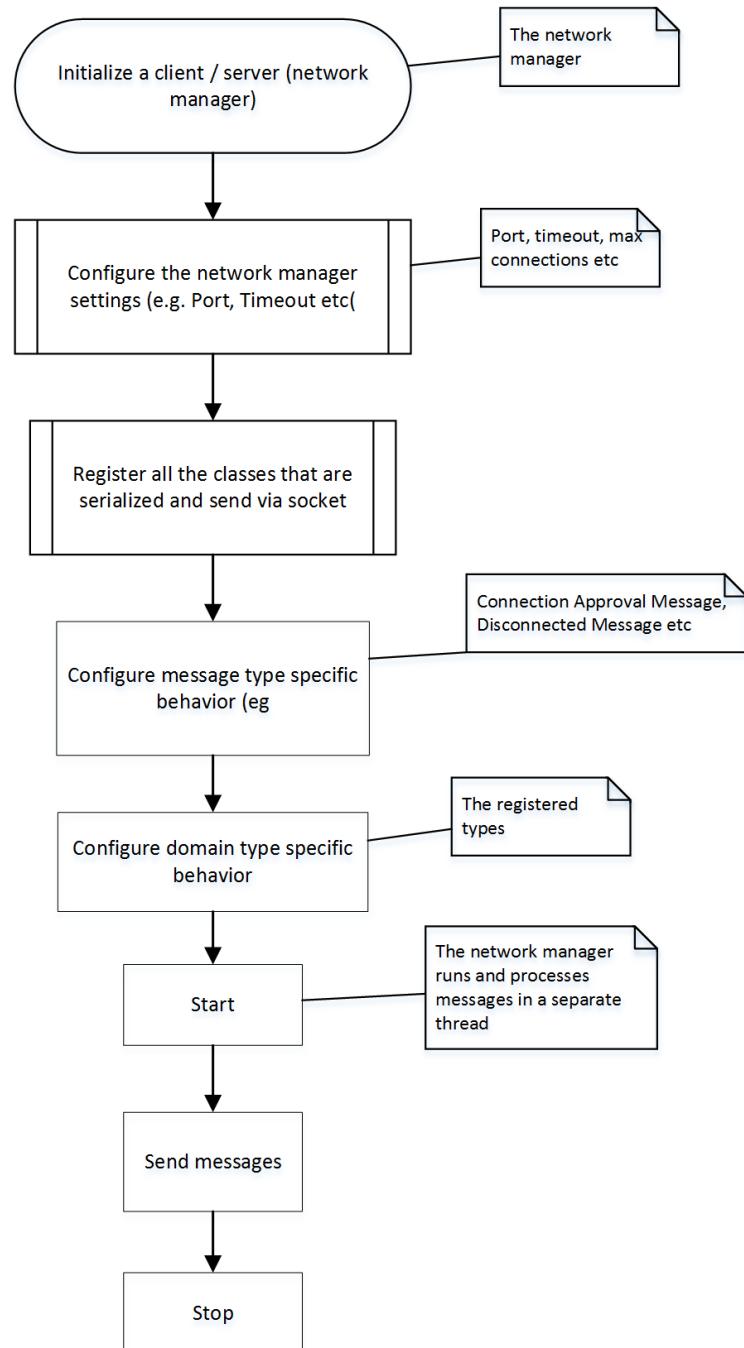
4.4.2 Αρχιτεκτονική

Η αρχιτεκτονική στηρίζεται στην ανταλλαγή μηνυμάτων-κλάσεων και χωρίζεται στα παρακάτω επίπεδα.

- Packages Χειρίζεται το serialization, αναλύει τα εισερχόμενα μηνύματα και χειρίζεται τις ανακατευθύνσεις του εκτελούμενου κώδικα κατά την παραλαβή μηνυμάτων.
- Networking Διαχειρίζεται τις συνδεσεις των χρηστών, τα sockets και ο,τι έχει να κάνει με διασύνδεση.
- Auditing Αφαιρετικό επίπεδο για logging. Ο χρήστης μπορεί να ενσωματώσει τη δική του λογική παρακολούθησης μηνυμάτων.
- Infrastructure Το επίπεδο αυτό περιέχει επαναχρησιμοποιήσημο κώδικα των πακέτων όπως το ConcurrentRepository, για thread-safe αποθήκευση κλειδιών και τιμών, και το ParallelBufferBlock το οποίο εκτελεί κώδικα σε buffer άλλου thread για ασύγχρονη επεξεργασία.

4.4.3 Packages Module

Το serialization των μηνυμάτων πρέπει να είναι ντερεμενιστικό ώστε να αναγνωρίζεται ο τύπος του μηνύματος και η κλάση την οποία αντιπροσωπεύει και ελαφρύς ώστε να μην επιβαρύνεται το network με επιπλέον δεδομένα. Ο ενσωματομένος serializer χρησιμοποιεί reflection για να αναγνωρίσει τα primitive properties της κλάσης. Για προχωριμένο serialization περίπλοκων τύπων, ο χρήστης έχει τη δυνατότητα να συμπεριλάβει τον δικό του serializer με το implementation του IPackageSerializer



Διάγραμμα 4.2: Network Usage Diagram

interface και την χωρίγηση του σημειώνοντας την κλάση με το PackageSerializerAttribute και τον τύπο του serializer.

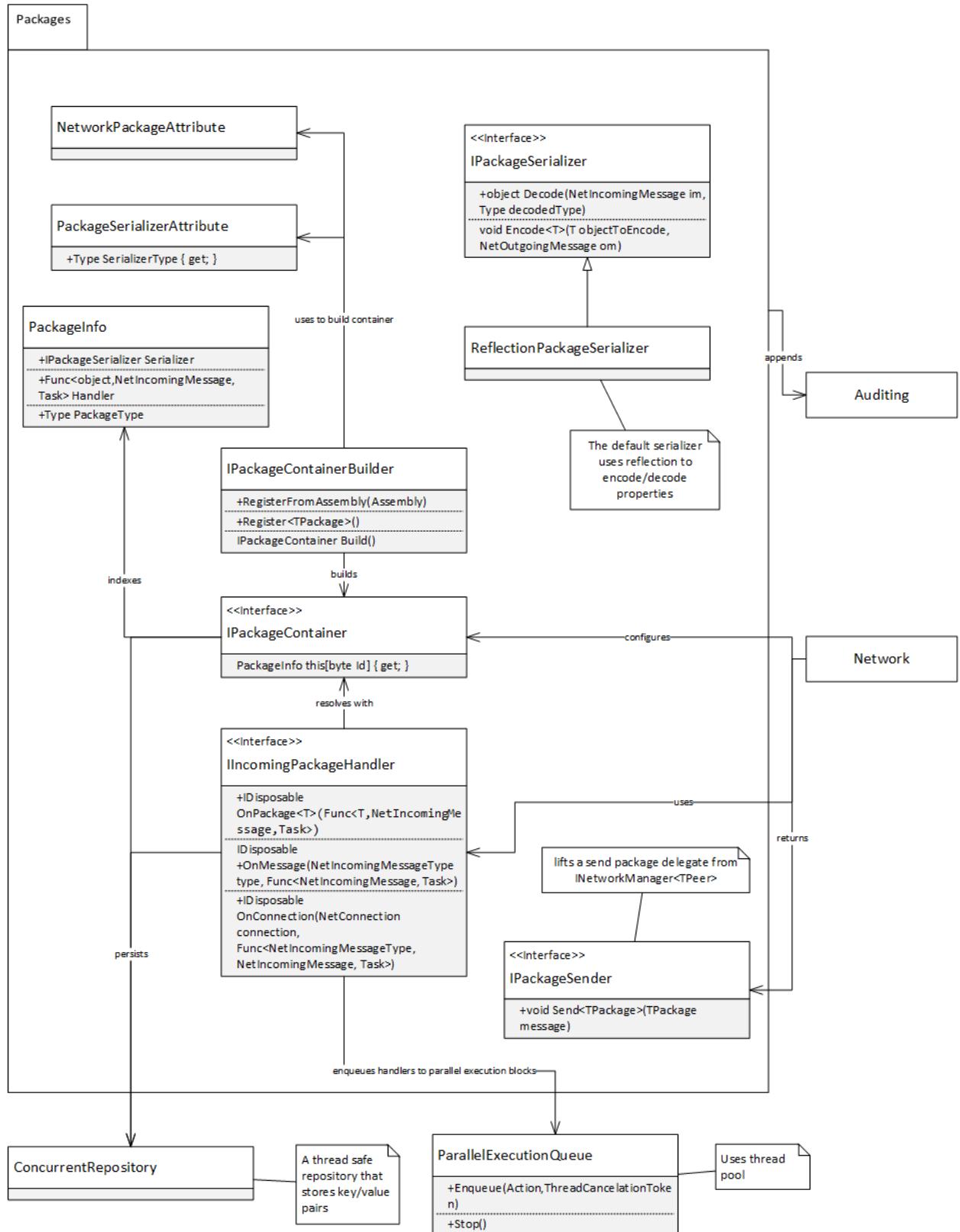
Κατά την προετοιμασία του network manager, ο χρήστης καλείται να καταχωρίσει στο Container ποιες κλασεις θα χρησιμοποιηθούν κατά την ανταλλαγή μηνυμάτων. Οι κλασεις που προωρίζονται για serialization σημειώνονται με κάποιο Attribute και η καταχώριση μπορεί να γίνει δυναμικά με reflection. Όταν τελειώσει η καταχώριση, το container χρησιμοποιώντας ένα ντετερμενιστικό αλγόριθμο ταξινομώντας με βάση το όνομα της κλάσης στο χώρο ονομάτων κτίζει ένα thread safe repository στο οποίο καταχωρείται ένα μοναδικό byte για αναγνωριστικό του κάθε τύπου και πληροφορίες για την χρήση του τύπου.

Στο τέλος της καταχώρισης και της κατασκευής αλγόριθμου αντιστοίχισης αντικειμένων και λογικής, ο χρήστης μπορεί να καταχωρίσει εκτελέσημο κώδικα υπό μορφή function ο οποίος εκτελείται ασύγχρονα κατά την παραλαβή κάποιου μηνύματος-κλάσης, τύπου μηνύματος, ή συμβάντος συγκεκριμένης σύνδεσης.

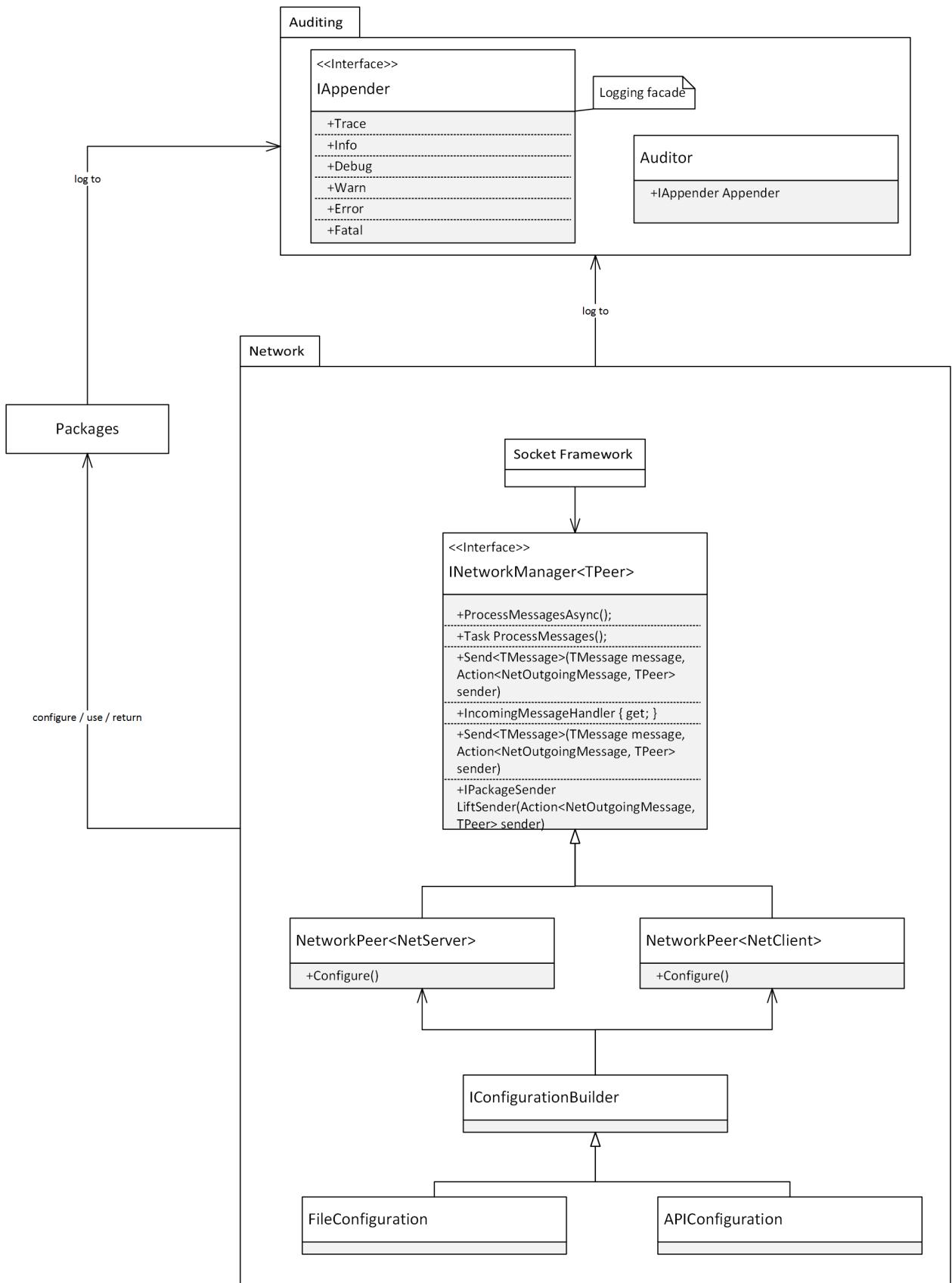
Στο UML διάγραμμα 4.3 παρουσιάζεται το υποσύστημα διαχείρισης πακέτων.

4.4.4 Networking Module

Οι ρυθμίσεις του network manager κτίζονται από αφαιρέσεις. Η χρήση πρωτοκόλλων και τεχνικών γίνεται με άγνοια της υλοποίησης και λεπτομερειών. Η σχεδίαση επιτρέπει την αποστολή μηνύματος ανεξαρτήτου πρωτοκόλου, τρόπο αποστολής και παραλήπτη. Ο network manager περιλαμβάνει τεχνικές function lifting για αποστολή μηνυμάτων. Ο χρήστης μπορεί να ρυθμίσει διάφορους τρόπους αποστολής ανάλογα με το περιβάλλον χρήση, και να τους κρύψει πίσω από functions. Η αρχιτεκτονική του networking module παρουσιάζεται στο uml διάγραμμα 4.4



Διάγραμμα 4.3: Network Packages Subsystem



Διάγραμμα 4.4: Networking Module

4.4.5 API

Συντομη επισκόπηση του API.

- Κλάσεις που προορίζονται για ανταλλαγή μηνυμάτων

Παράδειγμα κώδικα 4.1: Package Attribute

```

1 [ GinetPackage ]
2   // optional
3 [ PackageSerializer( typeof( MyCustomSerializer ) ) ]
4 public class MyPackage
5 {
6   public string Message { get; set; }
7 }
```

- Ρύθμιση server

Παράδειγμα κώδικα 4.2: Ρύθμιση server

```

1 var server = new NetworkServer("MyServer",
2 builder =>
3 {
4   // via reflection
5   builder.RegisterPackages(
6     Assembly.Load("Packages.Assembly.Name"));
7   // or manually
8   builder.RegisterPackage<MyPackage>();
9 }
10 cfg =>
11 {
12   cfg.Port = 1234;
13   cfg.ConnectionTimeout = 5.0f;
14   cfg.MaxConnections = 10;
15   // Additional configuration
16 }
17 // optional, logging output, default is the one supplied
18 out: new ActionAppender(Console.WriteLine)
```

- Καταγραφή κινήσεων

Παράδειγμα κώδικα 4.3: Καταγραφή κινήσεων

```
1 server.IncomingMessageHandler.LogTraffic();
```

- Απάντηση κατά την παραλαβή μηνύματος

Παράδειγμα κώδικα 4.4: Απάντηση κατά την παραλαβή μηνύματος

```
1 server.Broadcast<ChatMessage>((msg, im, om) =>
2 {
3     server.Out.Info($"Received {msg.Message}");
4     server.SendToAllExcept(om, im.SenderConnection,
5                             NetDeliveryMethod.ReliableOrdered, channel: 0);
6 },
7 packageTransformer: msg =>
8 msg.Message += " this is broadcasted");
9
10 server.BroadcastExceptSender<ChatMessage>((sender, msg) =>
11 {
12     server.Out.Info($"Broadcasting {msg.Message} . .
13 Received from: {sender}");
14});
```

- Κώδικας ο οποίος εκτελείται κατά την παραλαβή συγκεκριμένου τύπου μηνύματος

Παράδειγμα κώδικα 4.5: Incoming type handler

```
1 server.IncomingMessageHandler.OnMessage(
2     NetIncomingMessageType.ConnectionApproval,
3     incomingMsg =>
4 {
5     // Configure connection approval
6     var parsedMsg = server
7         .ReadAs<ConnectionApprovalMessage>(
8             incomingMsg);
```

```

8     if (parsedMsg.Password ==
9             "my\u00a0secret\u00a0and\u00a0encrypted\u00a0password")
10    {
11        incomingMsg.SenderConnection.Approve();
12        incomingMsg.SenderConnection.Tag =
13            parsedMsg.Sender;
14    }
15    else
16    {
17        incomingMsg.SenderConnection.Deny();
18    }
19 });

```

- Εκτελέσιμος κώδικας κατά την παραλαβή συγκεκριμένου μηνύματος-κλάσης

Παράδειγμα κώδικα 4.6: Incoming message handler

```

1 server.IncomingMessageHandler
2 .OnPackage<MyPackage>((msg, sender) =>
3     Console.WriteLine(
4         $"'{msg.Message}\u00a0from\u00a0{sender.SenderConnection}'"));

```

- Αποστολή μηνύματος-κλάσης

Παράδειγμα κώδικα 4.7: Incoming message handler

```

1 server.Send(
2     new MyPackage { Message = "Hello" },
3     (outgoingMessage, peer) =>
4         peer.SendMessage(outgoingMessage,
5                           NetDeliveryMethod.ReliableOrdered);

```

- Lift αποστολής μηνύματος

Παράδειγμα κώδικα 4.8: Message sender lift

```

1 var packageSender = client.LiftSender((msg, peer) =>
2     peer.SendMessage(msg,

```

```

3     NetDeliveryMethod . ReliableOrdered )) ;
4
5 packageSender . Send ( new MyPackage { Message = "Hello" } );

```

- Διαγραφή handler

Παράδειγμα κώδικα 4.9: Διαγραφή handler

```

1 var handlerDisposable = server . IncomingMessageHandler
2     . OnPackage<MyPackage>((msg, sender) => {});
3
4 handlerDisposable . Dispose ();

```

- Η ρύθμιση και το API του client είναι πανομοιότυπο με του server. Στη θέση του *NetworkServer* χρησιμοποιείται ο *NetworkClient* όπου και τα δύο υλοποιούν τον *NetworkManager<TPeer> where TPeer:NetPeer*

4.5 Testing

Μια καλή αρχιτεκονική υποστηρίζεται από την ευκολία της δοκιμαστικότητας ανά μονάδα (unit testability). To framework χρησιμοποιεί functional τεχνικές, οι οποίες εύκολα μπορούν να αντικατασταθούν με mocks. Ο χρήστης μπορεί να απομιμήσει την παραλαβή ή αποστολή πακέτων και να ενημερώνει την μονάδα επεξεργασίας μηνυμάτων από το τρέχων thread.

Παράδειγμα κώδικα 4.10: Incoming Package Handler

```

1
2 [TestMethod]
3 public void Incoming_Package_GetsHandled()
4 {
5     //Arrange
6     var client = new NetworkClient(
7         builder =>
8             builder . Register<MyPackage>(),
9             cfg=> {});

```

```

10
11     var mockedHandler =
12         new Mock<Func<MyPackage , NetIncomingMessage>>()
13             ;
14     mockedHandler . Setup( x =>
15         x( It . IsAny<MyPackage>() ) );
16
17     client
18     . IncomingMessageHandler
19     . OnPackage<MyPackage>( mockedHandler );
20
21     //Act
22     client . IncomingMessageHandler . SimulateReceive(
23         new MyPackage() );
24     client . ProcessMessages() . RunSynchronously();
25
26     //Assert
27     mockedHandler . Verify( x=>x() , Times . Once() );
}

```

Επίσης παρέχεται η δυνατότητα της προσομοίωσης χαμένων πακέτων, για να δοκιμαστεί πως η εφαρμογή ανταποκρίνεται σε περιβαλλον κακής διαδικτύωσης και να αναπτυχθούν τεχνικές network prediction για εξομάλυνση της κίνησης.

Κεφάλαιο 5

Διαγνωστικά και Αποσφαλμάτωση

Η ανάπτυξη του λογισμικού πρέπει να παρακολουθείται. Σε λογισμικά όπως ένα παιχνίδι, η επίδοση της εφαρμογής είναι εξαιρετικά σημαντική. Για τη μετάβαση από δοκιμαστικές σε τελικές εκδόσεις, χρειάζεται συλλογή και ανάλυση διαγνωστικών επιδόσεων για διαφορετικές πλατφόρμες, επεξεργαστές και κάρτες γραφικών. Ακομή και στις τελικές εκδόσεις όμως υπάρχει η πιθανότητα σφάλματος.

5.1 Logging and Tracing

Ένα logfile αρχείο περιέχει γεγονότα τα οποία συμβαίνουν κατά την εκτέλεση ενός λογισμικού. Logging ονομάζουμε τη διαδικασία καταγραφής μηνυμάτων στο logfile ανάλογα με το επίπεδο σημαντικότητας. Ένα εξειδικευμένο είδος logging είναι το tracing, στο οποίο καταγράφονται πληροφορίες για την εκτέλεση του προγράμματος. Χρησιμοποιείται κυρίως κατά την αποσφαλμάτωση αλλά και για συλλογή πληροφοριών με σκοπό την βελτίωση του λογισμικού για καλύτερη εμπειρία χρήστη.

Όταν συμβεί ένα κρίσιμο σφάλμα στο οποίο το λογισμικό τερματίζει, πρέπει να δημιουργηθεί ένα crash report και να σταλεί στους προγραμματιστές για να εντοπίσουν το σφάλμα. To crash report πρέπει να περιλαμβάνει:

- Το στιγμιότυπο του χρόνου σε UTC.
- Το επίπεδο στο οποίο παρουσιάστηκε το σφάλμα
- Η τοποθεσία του παίχτη

- Η κατάσταση του παιχτη
- Gameplay scripts
- Exception, stack trace
- Η κατάσταση κατανομής μνήμης
- Στιγμιότυπο οθόνης

5.2 Time Benchmarker

Ο time ruler προσφέρει δυναμική ανάλυση του χρόνου εκτέλεσης και των ενημερώσεων με απόδοση πραγματικού χρόνου στην οθόνη με φιλικό όνομα και χρώμα μεταξύ της έκτασης υπολογισμού του. Ο time ruler παίρνει στιγμιότυπα του χρόνου κατά το χρόνο εκτέλεσης και τα αποδίδει το μέσο όρο ανά δευτερόλεπτο στην οθόνη με τη μορφή γραμμής προόδου. Η απόδοση και τα πλαίσια υπολογισμού είναι τροποποιήσιμα και ελεγχόμενα από το σύστημα διαγνωστικών για εύκολη αναλλαγή.

Παράδειγμα κώδικα 5.1: Time Ruler

```

1 string aiRuler = "Enemy_AI_Pathfinding";
2 diagnostics.AddTimeRuler(rulerId, Color.Red);
3 diagnostics.TimeRuler[aiRuler].Show = true;
4
5 using(diagnostics.TimeRuler[aiRuler].Benchmarker)
6 {
7     //benchmarking logic here
8 }
```

5.3 Debug Drawing API

Μία οντοτητα η οποία αποδίδεται στην οθόνη περιέχει περισσότερες πληροφορίες από το αυτό που δείχνει. Μια οντότητα αν συμμετέχει στη προσομοίωση φυσικής έχει ταχύτητα και μάζα, αν συμμετέχει στο σύστημα συγκρούσεων έχει κουτί σύγκρουσης

το οποίο δείχνει τα όρια σύγκρουσης με άλλα αντικείμενα και πολλές άλλες πληροφορίες σχετικά με την κατάσταση στο παιχνίδι. Το υποσύστημα των diagnostics παρέχει λειτουργίες αποσφαλμάτωσης με απόδοση στην οθόνη και να είναι προσβάσιμο από τα υπόλοιπα συστήματα. Ο κώδικας αυτού του συστήματος παραλείπεται στα release builds. Αυτό επιτυχνάνεται με τη χρήση των conditionals τα οποία χρησιμοποιούν οδηγίες προεπεξεργαστή.

Το υποσύστημα προσφέρει απόδοση για:

- Βασικά σχήματα
- Σημεία
- Σφαίρες
- Άξονες συντεταγμένων
- Κουτιά οριοθέτησης
- Formatted text
- Δυνατότητα εναλλαγής χρωμάτων

5.4 Υποσυστήματα αποσφαλμάτωσης

Το σύστημα αποσφαλμάτωσης και διαγνωστικών περιέχει πολλά υποσυστήματα.

- Μενού επιλογών στο παιχνίδι με δυνατότητα εναλλαγής επιλογών και τροποποίησης τιμών.
- Debug camera η οποία κινείται χωρίς περιορισμούς στο χώρο.
- Κονσόλα η οποία εκτελεί scripts και εντολές υπό τη μορφή κειμένου παρόμοια με το Unix Shell.
- Assertions εντολές οι οποίες αν δεν αξιολογηθούν ως αληθές κατά το χρόνο εκτέλεσης του λογισμικού, τερματίζουν το λογισμικό με κωδικό σφάλματος.
- FPS Counter το οποίο αποδίδει στην οθόνη τον αριθμό των fps.

- Δυνατότητα παύσης.
- Cheats.
- Screenshots και screen captures.
- Ingame profiler: profiling blocks με αναγνώσιμα ονόματα.
- Ιεραρχικό Profiling.
- Εξαγωγή εδομένων σε μορφή η οποία είναι εύκολα αναγνώσιμη από άνθρωπο για αποσφαλμάτωση.
- Στατιστικά μνήμης και ανίχνευση διαρροών.

Κεφάλαιο 6

Επίλογος

Ο σχεδιασμός ενός λογισμικού το οποίο παράγει λογισμικό είναι πολύ σύνθετος και αφηρημένος. Η υλοποίηση των υπουποτημάτων των οποίων αναφέρθηκαν στηρίζεται σε γνώση εις βάθους διάφορων τεχνολογιών όπως OpenGL, SDL κλκ καθώς και σε γνώση ιδιαιτεροτήτων πλατφορμών, καρτών γραφικών κλπ. Κάποια από τα υπουποτήματα τα οποία δεν αναφέρθηκαν είναι τα παρακάτω:

- Υποσύστημα διαχείρισης ήχου
- Gameplay System το οποίο περιλαμβάνει τις λειτουργίες των μηχανισμών του παιχνιδιού και βρίσκεται στην κατηγορία των game-specific υπουποτημάτων.
- Camera System για το χειρισμό κάμερας.
- Rendering system. Βασίζεται σε καλή κατανόηση μαθηματικών, των σταδιών της κάρτας γραφικών (rendering pipeline) και σε πολύ χαμηλού επιπέδου βελτιστοποιήσεις για τη βέλτιστη χρήση του κύκλου του επεξεργαστή, της μνήμης και της κάρτας γραφικών.
- AI Το οποίο μπορεί περιλαμβάνει από απλά finite state machines και graph path finding algorithms όπως A*, μέχρι machine learning.

6.1 Επεκτασιμότητα

6.1.1 Plugins

Ως plug-in, ορίζεται ένα σύστημα συστατικών κάποιου λογισμικού που προσθέτει ιδιαίτερες δυνατότητες σε ένα μεγαλύτερο λογισμικό. Ένα εξειδικευμένο είδος plug-in είναι το add-on και περιλαμβάνει επεκτάσεις ή οπτικά θέματα.

Γιατί:

- Δυνατότητα άλλων προγραμματιστών να προγραμματίσουν επιπλέον δυνατότητες κάποιας εφαρμογής
- Υποστήριξη εύκολης πρόσθεσης νέων χαρακτηριστικών
- Μείωση του μεγέθους του πυρήνα μιας εφαρμογής
- Διαχωρισμός του πηγαίου κώδικα από της εφαρμογή σε περίπτωση ασύμβατων αδειών.

Επέκταση του Gem IDE μπορεί να γίνει χρησιμοποιώντας το Gem IDE Core και με εξαγωγεί τη βιβλιοθήκης σαν Module. Παράδειγμα plugin είναι το Gem.IDE.Modules.Spritesheets το οποίο οπτικοποιεί διαδικασία δημιουργίας και εξαγωγής 2D animation spritesheets.

6.2 Συμπεράματα

Βιβλιογραφία

- [1] Christer Ericson. *Real-Time Collision Detection*. CRC Press, Inc., Boca Raton, FL, USA, 2004.
- [2] J. Gregory. *Game Engine Architecture*. Ak Peters Series. Taylor & Francis, 2009.
- [3] Raph Koster. *A Theory of Fun for Game Design*. Paraglyph Press, 2004.
- [4] George Michaelides. Gem engine.
- [5] George Michaelides. Gem engine wiki.
- [6] George Michaelides. Ginet.
- [7] Robert Zubek Robin Hunicke, Marc LeBlanc. Mda: A formal approach to game design and game research. Game Design and Tuning Workshop at the Game Developers Conference, 2004.
- [8] Jaroslav Tulach. *Practical API Design: Confessions of a Java Framework Architect*. Apress, 2008.

Γλωσσάρι

API Η Διεπαφή Προγραμματισμού Εφαρμογών (αγγλ. API, από το Application Programming Interface), γνωστή και ως Διασύνδεση Προγραμματισμού Εφαρμογών (για συντομία διεπαφή ή διασύνδεση), είναι η διεπαφή των προγραμματιστικών διαδικασιών που παρέχει ένα λειτουργικό σύστημα, βιβλιοθήκη ή εφαρμογή προκειμένου να επιτρέπει να γίνονται προς αυτά αιτήσεις από άλλα προγράμματα ή/και ανταλλαγή δεδομένων.. 8, 24, 42

B-Tree In computer science, a B-tree is a self-balancing tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time. . 39

dpi Dots per inch (DPI, or dpi)[1] is a measure of spatial printing or video dot density, in particular the number of individual dots that can be placed in a line within the span of 1 inch (2.54 cm).. 39

fps Frame rate, also known as frame frequency, is the frequency (rate) at which an imaging device displays consecutive images called frames. The term applies equally to film and video cameras, computer graphics, and motion capture systems. Frame rate is expressed in frames per second (FPS).. 25, 26, 71

gui Γραφικό περιβάλλον χρήστη ή γραφική διασύνδεση/διεπαφή χρήστη (αγγλικά: Graphical User Interface, GUI) καλείται στην πληροφορική ένα σύνολο γραφικών στοιχείων, τα οποία εμφανίζονται στην οθόνη κάποιας ψηφιακής συσκευής (π.χ. Η/Υ) και χρησιμοποιούνται για την αλληλεπίδραση του χρήστη με τη συσκευή αυτή. Παρέχουν στον τελευταίο, μέσω γραφικών, ενδείξεις και εργαλεία προκειμένου αυτός να φέρει εις πέρας κάποιες επιθυμητές λειτουργίες. Για τον

λόγο αυτό δέχονται και είσοδο από τον χρήστη και αντιδρούν ανάλογα στα συμβάντα που αυτός προκαλεί με τη βοήθεια κάποιας συσκευής εισόδου (π.χ. πληκτρολόγιο, ποντίκι).. 27

MDA MDA is a formal approach to understanding games ñ one which attempts to bridge the gap between game design and development, game criticism, and technical game research. We believe this methodology will clarify and strengthen the iterative processes of developers, scholars and researchers alike, making it easier for all parties to decompose, study and design a broad class of game designs and game artifacts. . 21

Κατάλογος διαγραμμάτων

| | | |
|------|---|----|
| 2.1 | Τυπική αρχιτεκτονική μηχανής γραφικών | 17 |
| 3.1 | Κύκλος ζωής του πυρήνα | 25 |
| 3.2 | Game loop | 26 |
| 3.3 | screensystem sequence | 28 |
| 3.4 | Screen System UML | 30 |
| 3.5 | Input System UML | 34 |
| 3.6 | Physics System | 39 |
| 3.7 | UI B-Tree | 40 |
| 3.8 | UI Structure | 41 |
| 3.9 | UI System UML | 43 |
| 3.10 | UI Usage Diagram | 44 |
| 3.11 | Runtime Resource UML | 47 |
| 3.12 | Behavior Trees | 49 |
| 3.13 | State Machine UML | 51 |
| 3.14 | State Machine Example | 52 |
| 4.1 | Network Sequence Diagram | 55 |
| 4.2 | Network Usage Diagram | 60 |
| 4.3 | Network Packages Subsystem | 62 |
| 4.4 | Networking Module | 63 |

Κατάλογος παραδειγμάτων κώδικα

| | | |
|------|--|----|
| 3.1 | Transition Delegate | 29 |
| 3.2 | Transition Delegate | 31 |
| 3.3 | Παράδειγμα καταχώρησης listeners | 34 |
| 3.4 | Behavior Tree Builder | 50 |
| 3.5 | State machine | 52 |
| 4.1 | Package Attribute | 64 |
| 4.2 | Ρύθμιση server | 64 |
| 4.3 | Καταγραφή κινήσεων | 65 |
| 4.4 | Απάντηση κατά την παραλαβή μηνύματος | 65 |
| 4.5 | Incoming type handler | 65 |
| 4.6 | Incoming message handler | 66 |
| 4.7 | Incoming message handler | 66 |
| 4.8 | Message sender lift | 66 |
| 4.9 | Διαγραφή handler | 67 |
| 4.10 | Incoming Package Handler | 67 |
| 5.1 | Time Ruler | 70 |

Η εργασία αυτή στοιχειοθετήθηκε με το πρόγραμμα **X_EΛΑΤΕΧ**. Για τη στοιχειοθέτηση της βιβλιογραφίας χρησιμοποιήθηκε το πρόγραμμα **bib_Te_X**. Οι γραμματοσειρές που χρησιμοποιήθηκαν είναι οι **Times New Roman** και **Courier New**.