

**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 (hardware).

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 και πιο σύνθετος και περίπλοκος. Τα έργα γίνονται όλο και 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 σουνίτα 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

Η πτυχιακή μου εργασία είναι αφιερωμένη στους γονείς μου Χαράλαμπο και Παναγιώτα και τον αφελφό μου Κώστα για την στήριξη και υπομονή τους κατά τη διάρκεια των φοιτητικών μου χρόνων. Επίσης ευχαριστώ θερμά τον επιβλέποντα καθηγητή μου, κύριο Νίκο Πεταλίδη, για όλη τη βοήθεια, καθοδήγηση που μου προσέφερε και τη γνώση που μου μετέφερε. Τέλος ευχαριστώ τον συνεργάτη μου Δήμο Παύλου για την εμπειρία που απέκτησα μέσω της επαγγελματικής μας σταδιοδρομίας.

# Περιεχόμενα

<b>Υπεύθυνη δήλωση</b>	<b>2</b>
<b>Σύνοψη</b>	<b>3</b>
<b>Ευχαριστίες</b>	<b>4</b>
<b>1 Εισαγωγή</b>	<b>8</b>
1.1 Δομή της εργασίας . . . . .	9
1.2 Εργαλεία και βιβλιοθήκες . . . . .	9
1.3 Τεχνικές υλοποίησης . . . . .	11
1.4 Εργαλεία ανάπτυξης λογισμικού με τη βοήθεια υπολογιστή . . . . .	11
1.4.1 Κοινές λειτουργίες . . . . .	12
1.4.2 Πλεονεκτήματα της χρήσης του εργαλείου . . . . .	12
1.4.3 Χαρακτηριστικά ενός καλού εργαλείου . . . . .	13
<b>2 Ανάπτυξη ηλεκτρονικών παιχνιδιών</b>	<b>14</b>
2.0.4 Ιστορική αναδρομή . . . . .	14
2.1 Ανάπτυξη παιχνιδιών . . . . .	14
2.1.1 Μηχανές γραφικών . . . . .	15
2.1.2 Τι είναι μια μηχανή γραφικών . . . . .	15
2.1.3 Γιατί να χρησιμοποιήσει κάποιος μηχανή γραφικών; . . . . .	16
2.1.4 Δομή μιας μηχανής γραφικών . . . . .	16
2.2 Σχεδιασμός παιχνιδιών . . . . .	21
2.2.1 Μοντέλο MDA . . . . .	21
2.2.2 Πως ορίζεται ένα παιχνίδι . . . . .	22
2.3 Δομή μιας τυπικής ομάδας ανάπτυξης παιχνιδιών . . . . .	22

<b>3 Ο πυρήνας της Μηχανής</b>	<b>25</b>
3.1 Κύκλος ζωής πυρήνα . . . . .	25
3.1.1 Βρόγχος παιχνιδιού . . . . .	26
3.1.2 Υποσυστήματα . . . . .	28
3.1.3 Τεχνικές Ενημέρωσης Υποσυστημάτων . . . . .	28
3.2 Διαχείριση οθονών . . . . .	28
3.2.1 Απαιτήσεις συστήματος διαχείρισης σκηνής . . . . .	29
3.2.2 Συστατικά του συστήματος . . . . .	30
3.2.3 Αρχιτεκτονική . . . . .	31
3.2.4 Παραδείγματα χρήσης API . . . . .	32
3.3 Σύστημα εισόδων . . . . .	32
3.3.1 Συσκευές ανθρώπινης διεπαφής . . . . .	32
3.3.2 Τύποι εισόδων . . . . .	33
3.3.3 Απαιτήσεις του υποσυστήματος διαχείρισης εισόδων . . . . .	33
3.3.4 Ακροατές . . . . .	34
3.4 Φυσική και εντοπισμός συγκρούσεων . . . . .	36
3.4.1 Τα παιχνίδια ως soft real-time simulations . . . . .	36
3.4.2 Βασικές έννοιες φυσικής του συστήματος . . . . .	37
3.4.3 Οντότητες του συστήματος . . . . .	38
3.4.4 Αρχιτεκτονική . . . . .	39
3.5 Διεπαφή χρήστη . . . . .	39
3.5.1 Απαιτήσεις . . . . .	40
3.5.2 Δομή στη μνήμη . . . . .	41
3.5.3 Τα υποσυστήματα διεπαφής χρήστη . . . . .	42
3.5.4 Κύκλος ζωής διεπαφής χρήστη . . . . .	43
3.6 Διαχείριση πόρων . . . . .	44
3.6.1 Ευθύνες του offline resource manager . . . . .	46
3.6.2 Ευθύνες του runtime resource manager . . . . .	46
3.7 Τεχνητή νοημοσύνη . . . . .	48
3.7.1 Δέντρα συμπεριφορών . . . . .	49
3.7.2 Finite State Machines . . . . .	51

<b>4 Δικτύωση</b>	<b>55</b>
4.0.3 Αλληλεπίδραση σε εικονικό περιβάλλον . . . . .	55
4.0.4 Οι απαιτήσεις του υποσυστήματος δικτύωσης . . . . .	56
4.1 Ανάλυση εννοιών . . . . .	56
4.1.1 Πρωτόκολλο . . . . .	57
4.1.2 Αρχιτεκτονική δικτύου . . . . .	57
4.2 Σχεδίαση του υποσυστήματος . . . . .	58
4.2.1 Οντότητες . . . . .	58
4.2.2 Τύποι μηνυμάτων . . . . .	58
4.2.3 Διαχειριστής δικτύωσης . . . . .	59
4.3 Υλοποίηση . . . . .	60
4.3.1 Τρόπος χρήσης . . . . .	60
4.3.2 Αρχιτεκτονική . . . . .	60
4.3.3 Ανταλλαγή πακέτων . . . . .	60
4.3.4 Διαχείριση δικτύωσης . . . . .	62
4.3.5 Παραδείγματα API . . . . .	65
4.4 Testing . . . . .	68
<b>5 Διαγνωστικά και Αποσφαλμάτωση</b>	<b>70</b>
5.1 Logging and Tracing . . . . .	70
5.2 Time Benchmarking . . . . .	71
5.3 Debug Drawing API . . . . .	71
5.4 Υποσυστήματα αποσφαλμάτωσης . . . . .	72
<b>6 Επίλογος</b>	<b>74</b>
6.1 Επεκτασιμότητα . . . . .	74
6.2 Συμπεράσματα . . . . .	75

# Κεφάλαιο 1

## Εισαγωγή

**Σκοπός** Σκοπός της πτυχιακής είναι να εξηγήσει σε υψηλό επίπεδο τη σύνδεση και αλληλεπίδραση βασικών υποσυστημάτων 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.

- Τη βιβλιοθήκη *Gem Engine* η οποία περιέχει τις κύριες λειτουργίες και τα υποσυστήματα τα οποία απαρτίζουν τη μηχανή γραφικών όπως είναι, το σύστημα εισόδων και η διαχείριση οθονών.
- Το γραφικό περιβάλλον της μηχανής *Gem IDE* το οποίο προσφέρει οπτική αναπαράσταση των λειτουργιών του *Gem Engine*. Μέσω του *Gem IDE* ο χρήστης μπορεί να αποσφαλματώσει το παιχνίδι και να προσθέσει λειτουργίες μέσω γραφικού περιβάλλοντος.

Η υλοποίηση αξιοποιεί αντικειμενοστραφή (object-oriented) και functional τεχνικές, είναι γραμμένη σε C# και κατά την ανάπτυξη του πηγαίου κώδικα χρησιμοποιήθηκε το σύστημα ελέγχου αναθεωρήσεων git. Ο πηγαίος κώδικας βρίσκεται στο <http://www.github.com/gmich/gem> και στο <http://www.github.com/gmich/ginnet> και η τεκμηρίωσή του στο <http://www.github.com/gmich/gem/wiki>.

## 1.1 Δομή της εργασίας

Στο πρώτο κεφάλαιο γίνεται επεξήγηση των εννοιών και διαφορών της ανάπτυξης παιχνιδιών (game development) και του σχεδιασμού παιχνιδιών (game design) και ανάλυση μιας τυπικής ομάδας ανάπτυξης παιχνιδιών, στην οποία απευθύνεται το εργαλείο. Στο επόμενο κεφάλαιο γίνεται ανάλυση των βασικών υποσυστημάτων και λειτουργιών τα οποία βρίσκονται στον πυρήνα της βιβλιοθήκης, όπως η διαχείριση του κύκλου ζωής, το σύστημα εισόδων και η διαχείριση πόρων και είναι αναγκαία για τη λειτουργία της μηχανής. Κατά την ανάλυση γίνεται αναφορά στο πρόβλημα, επισκόπηση στην αρχιτεκτονική με διαγράμματα και περιπτώσεις χρήσεις μέσω του API. Στο τέταρτο κεφάλαιο αναλύεται το υποσύστημα δικτύωσης της μηχανής, το οποίο επιτρέπει την επικοινωνία μεταξύ πελατών (clients) μέσω sockets δικτύου (network sockets). Στο πέμπτο κεφάλαιο παρουσιάζονται τα υποσυστήματα αποσφαλμάτωσης της μηχανής μαζί με τεχνικές ελέγχου υγείας του λογισμικού, τόσο κατά τη διαδικασία δημιουργίας όσο και στις τελικές εκδόσεις. Στον επίλογο αναφέρονται οι τεχνικές επέκτασης της μηχανής με εισηγήσεις και συμπεράσματα.

## 1.2 Εργαλεία και βιβλιοθήκες

Με την διόγκωση των βιβλιοθηκών ανοικτού 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 ανακαλύψεις τον τροχό”. Η πτυχιακή χρησιμοποιεί διάφορα εργαλεία και βιβλιοθήκες τα οποία επιλύουν γενικά προβλήματα υποδομών λογισμικού (infrastructural) για να μπορεί να επικεντρωθεί σε επίπεδα που αφορούν τον τομέα ανάπτυξης παιχνιδιών. Τα βασικά εργαλεία τα οποία χρησιμοποιεί ή επεκτείνει

είναι τα παρακάτω:

- Monogame Βιβλιοθήκη για απόδοση (rendering) και ανάπτυξη σε πολλές πλατφόρμες (cross-platform) η οποία βρίσκεται ένα επίπεδο πάνω από τους οδηγούς (drivers) της κάρτας γραφικών.
- SDL Simple DirectMedia Layer μια βιβλιοθήκη για ανάπτυξη σε πολλές πλατφόρμες (cross-platform development library) η οποία χρησιμοποιείται για το σύστημα ήχου και εισόδου.
- TPL Ασύγχρονα πρότυπα για ταυτόχρονο και παράλληλο προγραμματισμό.
- Lidgren Αφαίρεση για sockets δικτύου (network sockets). Χρησιμοποιείται από το υποσύστημα δικτύωσης.
- Castle Proxy Βιβλιοθήκη για τροποποίηση κώδικα σε χαμηλότερο επίπεδο Common Intermediate Language.
- Farseer Βιβλιοθήκη προσομοίωσης φυσικού κόσμου. Χρησιμοποιείται από το υποσύστημα φυσικής και εντοπισμού συγκρούσεων.
- Autofac Inversion of Control. 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. Η προεπιλεγμένη βιβλιοθήκη στο υποσύστημα διαγνωστικών για δημοσίευση μηνυμάτων.
- Glsl Shaders. OpenGL Shading Language η οποία χρησιμοποιείται κατευθείαν στον αγωγό (pipeline) της κάρτας γραφικών. Χρησιμοποιείται για καλή επίδοση στη διαδικασία σκίασης, στις ανακλάσεις, εφέ (effects) και σε υπολογισμούς οι οποίοι είναι πολύ εντατικοί για τον επεξεργαστή.
- MEF Managent Extensibility Framework. Βιβλιοθήκη για δημιουργία λογισμικού με δυνατότητες δυναμικής επέκτασης και διαχείρισης plugins. Χρησιμοποιείται για να βρίσκει και να φορτώνει τα plugins του περιβάλλοντος ανάπτυξης.

- WPF Windows Presentation Foundation. Χρησιμοποιείται για τη δημιουργία του γραφικού περιβάλλοντος του περιβάλλοντος ανάπτυξης *Gem IDE*.
- 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 ενημέρωσή του.
- Moq Χρησιμοποιείται για δημιουργία ψεύτικων αντικειμένων με τροποποιήσιμη συμπεριφορά για δοκιμές.
- MsTest Εργαλείο δοκιμών μονάδας (unit tests) βασισμένο σε ισχυρισμούς (assertions).
- 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 αυστηρά σε SOLID principles. 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 τροποποιήσιμη συγκρότηση λόγω inversion of control.

### 1.4 Εργαλεία ανάπτυξης λογισμικού με τη βοήθεια υπολογιστή

Η διαδικασία ανάπτυξης λογισμικού είναι ακριβή και ο σχεδιασμός γίνεται όλο και πιο σύνθετος και περίπλοκος. Τα έργα γίνονται όλο και 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 δημοφιλή, 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 λειτουργιών ενός case tool δεν μπορεί να γίνει γιατί οι λειτουργίες διαφέρουν ανάλογα με τον σκοπό χρήσης. Οι λειτουργίες 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 Πλεονεκτήματα της χρήσης 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 κώδικα και της ροή δεδομένων για καλύτερη κατανόηση και ελέγχου του λογισμικού. Το εργαλείο αναλαμβάνει για την βέλτιστη χρήση των διαθέσιμων πόρων, ούτως ώστε η ομάδα ανάπτυξης να επικεντρωθεί στον πυρήνα της εφαρμογής τους. Η ανάλυση, η ανάπτυξη και ο σχεδιασμός γίνονται 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 οι 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 εργαλείου

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

- Τυποποιημένη μεθοδολογία χρησιμοποιώντας τεχνικές μοντελοποίησης όπως UML.
- Flexibility (ευελιξία): το εργαλείο πρέπει να προσφέρει στον χρήστη τη δυνατότητα να επιλέγει ποια εργαλεία να χρησιμοποιήσει.
- Strong integration (ισχυρή ένταξη): το εργαλείο πρέπει να υποστηρίζει όλα τα στάδια ανάπτυξης. Όταν γίνεται μια αλλαγή, τα στάδια τα οποία επηρεάζονται πρέπει να τροποποιούνται κατάλληλα.
- Ενσωμάτωση με εργαλεία ελέγχου.
- Reverse-engineering (αντίστροφη μηχανική): 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.

### 2.0.4 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 ζωή οι υπολογιστές, για την ακρίβεια, οι κεντρικοί υπολογιστές. Από εκείνη την περίοδο, τα βιντεοπαιχνίδια έκαναν την εμφάνισή τους, στις κονσόλες, στα φλίπερ, στους υπολογιστές, αλλά και στις φορητές κονσόλες. Από τότε η δημιουργία παιχνιδιών έχει γιγαντωθεί έχοντας ένα τεράστιο κομμάτι της παγκόσμιας οικονομίας. Πλέον ο ανταγωνισμός είναι τεράστιος, τα βιντεοπαιχνίδια κυκλοφορούν με πολύ γρήγορο ρυθμό, για διάφορες κονσόλες, με πολύ απαιτητικά γραφικά.

### 2.1 Ανάπτυξη παιχνιδιών

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

### 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 έχουν την ευχέρεια να επικεντρώνονται στις λεπτομέρειες του παιχνιδιού τους. Οι μηχανές αναπτύσσονται από ομάδες που απαρτίζονται όχι μόνο από προγραμματιστές, αλλά και από μαθηματικούς, φυσικούς κλπ. Η κάθε υποομάδα εστιάζει σε ένα συγκεκριμένο υποσύστημα. Για παράδειγμα οι φυσικοί ασχολούνται με τον εντοπισμό συγκρούσεων, ενώ οι αρχιτέκτονες λογισμικού ασχολούνται με τη σύνδεση και αλληλεπίδραση των υποσυστημάτων σε υψηλό επίπεδο, χωρίς να τους απασχολούν οι λεπτομέρειες υλοποίησης του κάθε υποσυστήματος.

### 2.1.2 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 τη απόδοση (rendering system), τον εντοπισμό συγκρούσεων (collision detection system), το υποσύστημα ήχου (audio system), το υποσύστημα διαχείρισης πόρων (resource management system) κλπ. Η αξία αυτού του διαχωρισμού εκτιμήθηκε από την κοινότητα όταν οι προγραμματιστές ξεκίνησαν να πουλάνε άδειες για το λογισμικό και επαναχρησιμοποιούσαν εργαλεία προηγούμενων παιχνιδιών με δημιουργία νέων assets. Μικρότερα στούντιο τροποποιούσαν εκδόσεις υπάρχοντων παιχνιδιών χρησιμοποιώντας το SDK. Πολλά παιχνίδια γράφτηκαν με σκοπό την επαναχρησιμοποίηση κώδικα και modding. Πολλές μηχανές, όπως η μηχανή του Quake III, γράφτηκαν με τρόπο ώστε να είναι εύκολα προσαρμόσιμες χρησιμοποιώντας scripting και με σκοπό την εμπορευματοποίηση μέσω αδειοδότησης

(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 η οδηγούμενη από δεδομένα αρχιτεκτονική (data-driven architecture) όπου οι κανόνες και τα στοιχεία δεν είναι σκληρά κωδικοποιημένα (hard coded), αλλά διαβάζονται από εξωτερικό αρχείο. Τα χαρακτηριστικά και τα όρια των μηχανών ορίζονται ανάλογα με το είδος του παιχνιδιού στο οποίο η μηχανή εστιάζει και με βάση τις πλατφόρμες και τις κάρτες γραφικών στις οποίες προορίζεται η ανάπτυξη. Για παράδειγμα η OpenGL ES η οποία χρησιμοποιείται από τα κινήτα είναι υποσύνολο της OpenGL και δεν υποστηρίζει όλες τις λειτουργίες της OpenGL [1].

### 2.1.3 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 απαλλάσσουν τους γραφίστες και τους προγραμματιστές από τις τεχνικές λεπτομέρειες, και εστιάζουν στην αισθητική και στο gameplay. Επίσης με την αποσύνδεση των συστημάτων έχουμε πιο προβλέψιμη συμπεριφορά, επεκτασιμότητα των υποσυστημάτων ως υποσυστήματα και εύκολη δοκιμαστικότητα (testability).

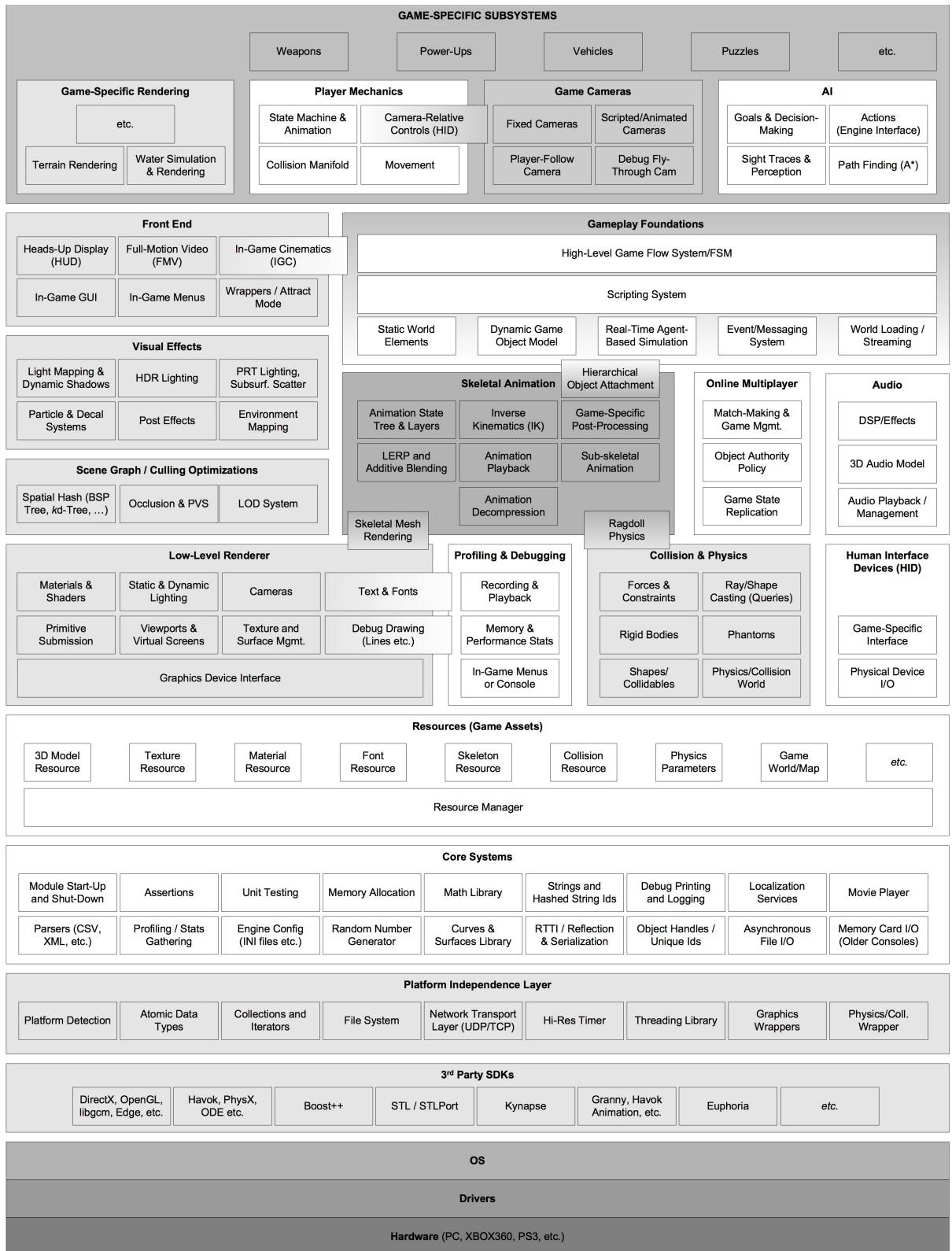
### 2.1.4 Δομή μιας μηχανής γραφικών

Η μοντελοποίηση 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 το σχεδιασμό συγκεκριμένου είδους παιχνιδιών. Μια τυπική μηχανή γραφικών παρουσιάζεται στην εικόνα 2.1 [8].

#### Επισκόπηση επιπέδων της μηχανής γραφικών

**Υλισμικό** Το υλισμικό (hardware) αντιπροσωπεύει το υλικό στο οποίο θα τρέξει το παιχνίδι, δηλαδή τον υπολογιστή ή την κονσόλα.

**Οδηγοί** Οι οδηγοί (drivers) διαχειρίζονται τους πόρους του υλικού και του λειτουργικού και παρέχουν ένα καθολικό πρωτόκολλο επικοινωνίας μεταξύ των διαφόρων παραλλαγών.



**Διάγραμμα 2.1:** Τυπική αρχιτεκτονική μηχανής γραφικών

**Λειτουργικό Σύστημα** Το λειτουργικό σύστημα είναι ο ενορχηστρωτής των προγραμμάτων. Κατανέμει το χρόνο μεταξύ των πόρων του υλικού και των προγραμμάτων και παραλληλίζει την εκτέλεση των προγραμμάτων. Στις κονσόλες το λειτουργικό παίζει τυπικό ρόλο, αφού το λογισμικό (παιχνίδια) έχει 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.

**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 δομές δεδομένων, η για απόδοση (rendering).

**Εντοπισμός συγκρούσεων** Εντοπισμός συγκρούσεων και δυναμική άκαμπτου σώματος (rigid-body dynamics) δηλαδή το πώς ένα σύστημα με διασυνδεδεμένα σώματα αντιδρά όταν του ασκούνται εξωτερικές δυνάμεις.

**Ανεξαρτησία πλατφόρμας** Πολλά παιχνίδια καλούνται να τρέξουν σε περισσότερες από μία πλατφόρμες. Στο επίπεδο ανεξαρτησίας πλατφόρμας (platform independence layer) οι εντολές του λειτουργικού συστήματος και υλικού ενθυλακώνονται (encapsulated) για να μπορούν να αλλάξουν ανάλογα με το περιβάλλον ανάπτυξης.

**Συστήματα Πυρήνα** Κάθε μεγάλο και σύνθετο πρόγραμμα χρειάζεται κάποιες βιβλιοθήκες γενικής χρήσης. Κάποιες από αυτές είναι:

**Ισχυρισμοί** Ισχυρισμούς (assertions) ονομάζουμε τα κομμάτια κώδικα, τα οποία ελέγχουν τις υποθέσεις του προγραμματιστή, για το πώς πρέπει να εκτελεστεί η συνάρτηση. Ο κώδικας αυτός αφαιρείται στις τελικές εκδόσεις, αφού περάσει η περίοδος δοκιμών.

**Διαχείριση μνήμης** Στο σύστημα διαχείρισης μνήμης (memory management) γίνεται έλεγχος της μνήμης για αποφυγή θρυμματισμού (fragmentation) και συνθήκες μη διαθέσιμης μνήμης (out of memory).

**Βιβλιοθήκη μαθηματικών** Η βιβλιοθήκη μαθηματικών περιέχει μαθηματικά,

τα οποία χρειάζονται στις προσομοιώσεις πραγματικού χρόνου, όπως είναι τα διανύσματα, οι πίνακες, η γεωμετρία κλπ.

**Διαχείριση πόρων** Το υποσύστημα διαχείρισης πόρων (resource management) ασχολείται με τη διαχείριση των πόρων του παιχνιδιού, όπως τα μοντέλα, τα textures, ο κόσμος, οι χάρτες κλπ.

**Μηχανή απόδοσης** Η μηχανή απόδοσης (rendering engine) είναι το πιο σύνθετο κομμάτι της μηχανής, το οποίο είναι υπεύθυνο για την αναπαράσταση του παιχνιδιού στην οθόνη.

**Απόδοση χαμηλού επιπέδου** Η απόδοση χαμηλού επιπέδου (low-level renderer) εστιάζει στην βελτιστοποίηση των πρωτογενών γεωμετρικών σχημάτων.

**Γράφος σκηνής** Ο γράφος σκηνής (scene graph) καθορίζει ποια αντικείμενα πρέπει να αποδοθούν (render) από τη μηχανή απόδοσης (rendering engine), χρησιμοποιώντας αλγόριθμους ανάλογα με το μέγεθος και το είδος του παιχνιδιού.

**Οπτικά εφέ** Τα οπτικά εφέ (visual effects) αποτελούνται από το σύστημα σωματιδίων (particle system), τη χαρτογράφηση φωτός (light mapping), τις δυναμικές σκιές (dynamic shadows), το anti-aliasing κλπ.

**Front End** Front end ονομάζεται το επίπεδο επικοινωνίας με το παιχνίδι, όπως το μενού, η κονσόλα, τα pop-ups παράθυρα κλπ.

**Profiling and Debugging** Η ποιότητα της απόδοσης του παιχνιδιού είναι πολύ κρίσιμη. Για να γίνει βελτιστοποίηση της απόδοσης, χρειάζονται εργαλεία ανάλυσης του υλικού στο οποίο τρέχει το παιχνίδι με οπτική αναπαράσταση.

**Σύστημα συγκρούσεων και φυσικής** Η φυσική και οι συγκρούσεις (physics and collision) είναι στενά συνδεδεμένες γιατί οι συγκρούσεις επιλύονται με κανόνες φυσικής.

**Animation** Animation είναι η ταχεία προβολή μιας σειράς από εικόνες (δισδιάστατης ή τρισδιάστατης μακέτας) ή θέσεων ενός μοντέλου, έτσι ώστε να δημιουργείται η ψευδαίσθηση της κίνησης.

**Συσκευές ανθρώπινης διεπαφής** Οι συσκευές ανθρώπινης διεπαφής (human interface devices) είναι 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.

**Ήχος** 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.

**Δικτύωση** Συνήθως ένας χρήστης παίζει μόνος του σε ένα εικονικό κόσμο. Πολλές φορές όμως άλλοι χρήστες συνδέονται σε αυτό τον κόσμο μέσω online multilayer.

**Gameplay Foundation Systems** Στο gameplay foundation system ορίζονται οι κανόνες του εικονικού κόσμου, οι οποίοι οριοθετούν τις δυνατότητες του παιχτη.

**Game Worlds and Object Models** Η αναπαράσταση του κόσμου με αντικειμενοστραφή τρόπο (game object model) περιλαμβάνει το στατικό background, τη δυναμική στερεών σωμάτων (dynamic rigid bodies), τους παίχτες (player characters), τους non-player characters, τα όπλα, τα οχήματα, τον φωτισμό, τις κάμερες κλπ.

**Σύστημα συμβάντων** Το σύστημα συμβάντων (event system) είναι το σύστημα το οποίο επιτρέπει την επικοινωνία του μοντέλου αντικειμένου παιχνιδιού (game object model) με τον εαυτό του.

**Scripting System** To scripting σε μία μηχανή γραφικών, επιταχύνει την ανάπτυξη γιατί περιέχει χρήσιμες, συχνές και έτοιμες προς εκτέλεση εντολές.

**Τεχνητή νοημοσύνη** Η τεχνητή νοημοσύνη (artificial intelligence) περιλαμβάνει τυπικά patterns νοημοσύνης, όπως την πλοιόγηση (navigation), την εύρεση διαδρομής (path finding), τη δυναμική αποφυγή αντικειμένων (dynamic object avoidance) κλπ.

**Game Specific Subsystems** Game specific subsystems ονομάζουμε τα συστήματα τα οποία βρίσκονται σε πιο ψηλά επίπεδα από τη μηχανή γραφικών και αφορούν το παιχνίδι το οποίο αναπτύσσεται.

**Tools and Asset Pipeline** Όλα τα παιχνίδια χρειάζονται πολλά δεδομένα για να αναπαραστήσουν τον εικονικό κόσμο, όπως τη διαμόρφωση (configuration), τα scripts, τα τρισδιάστατα μοντέλα κλπ. Οι μηχανές πρέπει να είναι σε θέση να επεξεργάζονται συγκεκριμένου τύπου δεδομένα τα οποία εξάγονται από δημοφιλή προγράμματα δημιουργίας ψηφιακού περιεχομένου ( digital content creation programs).

**Βάση δεδομένων πόρων** Η βάση δεδομένων πόρων (resource database) υλοποιεί τεχνικές αναζήτησης και αποθήκευσης των πόρων του παιχνιδιού. Χρησιμοποιούνται από υπάρχουσες 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 δημιουργίας παιχνιδιών (game design). Οι σχεδιαστές παιχνιδιών (game designers) σχεδιάζουν το πώς θα είναι το παιχνίδι χωρίς να τους απασχολούν οι τεχνικές λεπτομέρειες. Είναι υπεύθυνοι για:

- Τα εργαλεία και μηχανισμούς μέσα στο παιχνίδι
- Την ανάπτυξη κανόνων
- Την ιστορία και πλοκή
- Την στρατηγική και την 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 στη μηχανική (mechanics), δυναμική (dynamics) και αισθητική (aesthetics). [13].

**Μηχανική** Μηχανική ονομάζουμε τα διάφορα συστατικά ενός παιχνιδιού, στο επίπεδο αλγορίθμων και της αναπαράστασης.

**Δυναμική** Δυναμική είναι η συμπεριφορά των mechanics κατά τον χρόνο εκτέλεσης, αντιδρώντας στις εισόδους και εξόδους του συστήματος με την πάροδο του χρόνου

**Αισθητική** Αισθητική ονομάζουμε τις 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 παραγόντων οι οποίοι δρουν 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 [10], ορίζει το παιχνίδι ως μια διαδραστική εμπειρία, η οποία παρέχει στον χρήστη μια προκλητική σειρά από πρότυπα (patterns) τα οποία μαθαίνει και στην τελική εξειδικεύεται. Ισχυρίζεται ότι η εκμάθηση και εξειδίκευση βρίσκονται στην καρδιά του τι θεωρείται διασκεδαστικό, όπως ένα λογοπαίγνιο γίνεται αστείο τη στιγμή που ο εγκέφαλος αντιληφθεί το πρότυπο.

Στο βιβλίο The Grasshopper του Bernand Suits [14]. καθηγητή φιλοσοφίας στο πανεπιστήμιο του Waterloo, δηλώνει ότι ”ένα παιχνίδι είναι η εθελοντική προσπάθεια να ξεπεραστούν περιττά εμπόδια”. Ο Tracy Fullerton στο βιβλίο του Game Design Workshop [5] ορίζει ένα παιχνίδι ως ”ένα κλειστό, σύστημα που δεσμεύει τους παίκτες σε ένα δομημένο περιβάλλον σύγκρουσης και επιλύει την αβεβαιότητά του με ένα άνισο αποτέλεσμα. Οι ορισμοί είναι επηρεασμένοι από το περιβάλλον και τις προθέσεις των συγγραφέων, είναι όμως σωστοί με το δικό τους τρόπο.

## 2.3 Δομή μιας τυπικής ομάδας ανάπτυξης παιχνιδιών

Πριν από την ανάλυση της δομής της μηχανής, θα γίνει ανάλυση της δομής της ομάδας η οποία θα χρησιμοποιεί τη μηχανή, για να αναπτυχθούν στοχευμένα εργαλεία

για το κάθε πρόβλημα της κάθε υποομάδας.

**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 runtime programmers ασχολούνται με τη μηχανή και το παιχνίδι.

**Προγραμματιστές εργαλείων** Οι προγραμματιστές εργαλείων γράφουν εργαλεία τα οποία αυτοματοποιούν και ευκολύνουν τη διαδικασία ανάπτυξης.

Οι μηχανικοί έχουν είτε κάποια ειδικότητα, για παράδειγμα ειδικότητα στη τεχνητή νοημοσύνη, είτε είναι generalists, δηλαδή κατέχουν από όλα τα στοιχεία και μπορούν να λύσουν προβλήματα που προκύπτουν 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 concept artists 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 3D modelers είναι υπεύθυνοι για την τρισδιάστατη γεωμετρία του εικονικού κόσμου του παιχνιδιού. Απαρτίζονται από τους foreground modelers οι οποίοι σχεδιάζουν χαρακτήρες, οχήματα, όπλα και αντικείμενα του τρισδιάστατου κόσμου background modelers οι οποίοι σχεδιάζουν το στατικό περιβάλλον πχ κτήρια.

**Texture artists** Οι texture artists σχεδιάζουν τις δισδιάστατες εικόνες που καλύπτουν τα τρισδιάστατα μοντέλα

**Lighting artists** Οι lighting artists ορίζουν τις στατικές και δυναμικές πηγές φωτός και δουλεύουν με το χρώμα και την κατεύθυνση του φωτός.

**Animators** Οι animators σχεδιάζουν την κίνηση των χαρακτήρων και των αντικειμένων

**Ηθοποιοί σύλληψης κίνησης** Οι ηθοποιοί σύλληψης κίνησης (motion capture actors) παρέχουν ακατέργαστα δεδομένα κίνησης για να επεξεργαστούν οι animators και να τα ενσωματώσουν στο παιχνίδι.

**Sound designers** Οι sound designers παράγουν τα εφέ και τη μουσική.

**Voice actors** Η φωνή των voice actors 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 μοναδικά και αποφασίζουν για τη γεωμετρία στο περιβάλλον, πότε και πού εμφανίζονται χαρακτήρες και διάφορα αντικείμενα, πώς 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 μεγάλων εταιριών, όπου υπάρχουν πολλά τμήματα και πολλοί εργαζόμενοι.

## Κεφάλαιο 3

# Ο πυρήνας της Μηχανής

Μια βιβλιοθήκη (framework) η οποίο επεκτείνεται και διακλαδώνεται σε πολλές υποβιβλιοθήκες, στηρίζεται στη θεμελιώση ενός πηρύνα (framework core) ώστε να μπορέσει να επεκταθεί με σταθερότητα και συνέχεια. Το API του πυρήνα πρέπει να είναι εύκολο στην κατανόηση, να αποτελείται από αυτοεπεξηγούμενες αφαιρέσεις, οι οποίες εκθέτουν μόνο τα απολύτως απαραίτητα, ώστε οι υποβιβλιοθήκες που χρησιμοποιούν τον πυρήνα να μην δεσμεύονται σε υλοποιήσεις, να περιορίζει τον χρήστη σε συγκεκριμένα στάδια χρήσης για αποφυγή απρόβλεπτων συμπεριφορών και να προσφέρει δυνατότητες επεκτασιμότητας. [15] Ο πυρήνας περιέχει υποσυστήματα τα οποία μπορούν να χρησιμοποιηθούν τη δημιουργία διαφόρων ειδών παιχνιδιών.

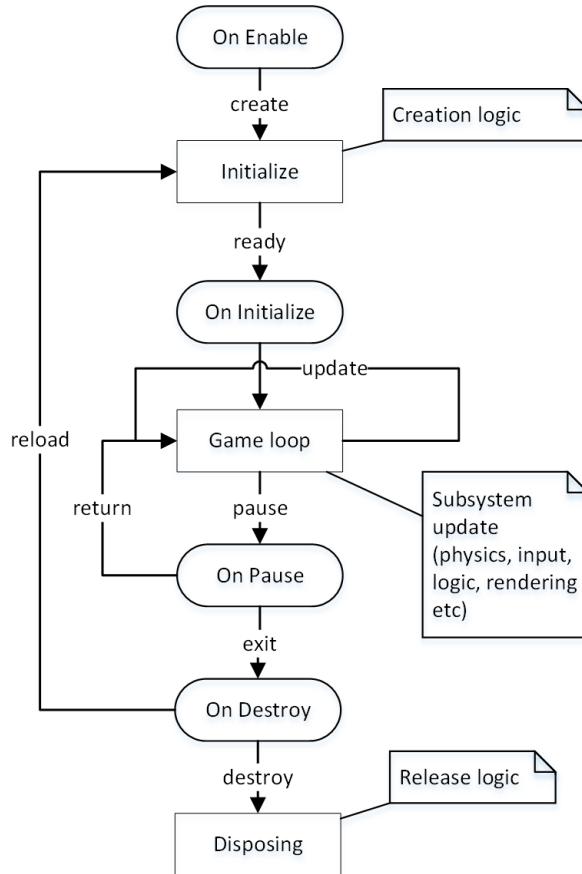
### 3.1 Κύκλος ζωής πυρήνα

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

**Initialization** Το initialization (προετοιμασία) εκτελείται μια φορά κατά τη δημιουργία του παιχνιδιού.

**Game loop** Το game loop (βρόγχος παιχνιδιού) εκτελείται με σταθερό χρονικό βήμα σε βρόγχο.

**Disposing** Το disposing εκτελείται μια φορά στην έξοδο.



**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 βρόγχο του παιχνιδιού ενημερώνονται όλα τα υποσυστήματα 3.2.

Το κάθε υποσύστημα έχει διαφορετικές απαιτήσεις για τη βέλτιστη και επιθυμητή λειτουργία. Το σύστημα συγκρούσεων χρειάζεται να ενημερώνεται εκατό φορές το δευτερόλεπτο, ενώ το σύστημα τεχνητής νοημοσύνης δύο φορές το δευτερόλεπτο [8]. Η πιο συνηθισμένη τεχνική είναι να υπάρχουν δύο κύριοι βρόγχοι ενημέρωσης (update loops):

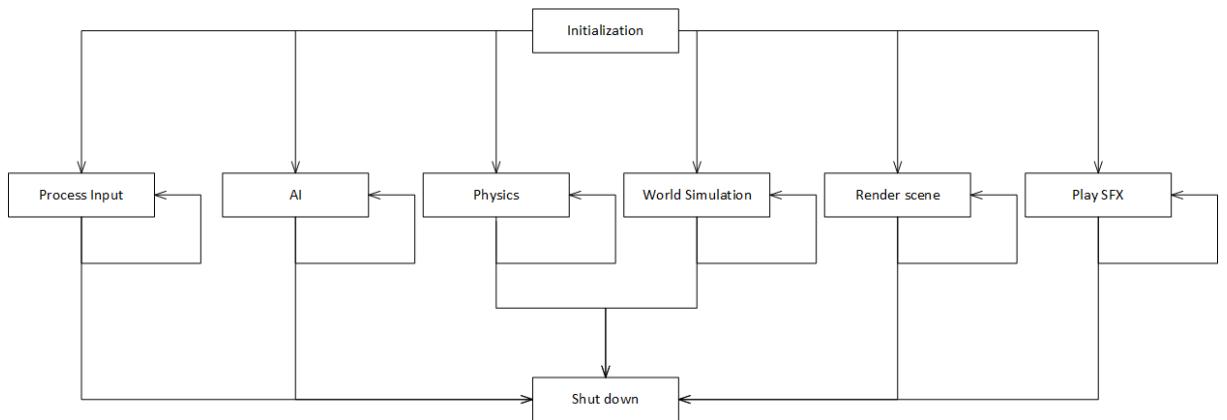
**Βρόγχος παιχνιδιού** Στο βρόγχο παιχνιδιού ενημερώνονται όλα τα υποσυστήματα όπως το υποσύστημα φυσικής, λογικής, δυναμικών κλπ.

**Rendering loop** Στο rendering loop (βρόχο απόδοσης) οποίο γίνονται οι κλήσεις στην κάρτα γραφικών και ενημερώνεται στις 50-60 φορές το δευτερόλεπτο με αποτέλεσμα η οθόνη να ενημερώνεται 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 η μηχανή να μετράει και να αποθηκεύει στη μνήμη τη διαφορά χρόνου μεταξύ κάθε κλήσης, να εκτελεί τον κώδικα στο συγκεκριμένο βρόχο και για τον υπόλοιπο χρόνο το νήμα (thread) στο οποίο εκτελείται η λογική ενημέρωσης του παιχνιδιού να κοιμάται (sleep).



**Διάγραμμα 3.2:** Βρόγχος παιχνιδιού

**Χρόνος** Η έννοια της διάρκειας και του χρόνου στα συστήματα πραγματικού χρόνου πρέπει να αντιμετωπίζεται απομονωμένα. Ο χρόνος του παιχνιδιού είναι ανεξάρτητος από τον πραγματικό χρόνο. Η ενημέρωση 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 αποδίδεται σε προγματικό χρόνο, μπορεί να αποδοθεί 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 αντιπροσωπεύει τη διαφορά χρόνου.

### 3.1.2 Υποσυστήματα

Το κάθε υποσύστημα δημιουργείται, ενημερώνεται και καταστρέφεται μέσα στο γενικό κύκλο ζωής του πηρύνα. 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 αν και απομονωμένα και χωρίς αλληλεξαρτήσεις πρέπει με κάποιο τρόπο να ενημερώνονται και να επικοινωνούν μεταξύ τους ή να στέλνουν μηνύματα όταν συμβεί κάποιο γεγονός. Οι τεχνικές ενημέρωσης υποσυστημάτων είναι οι παρακάτω:

**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 [4].

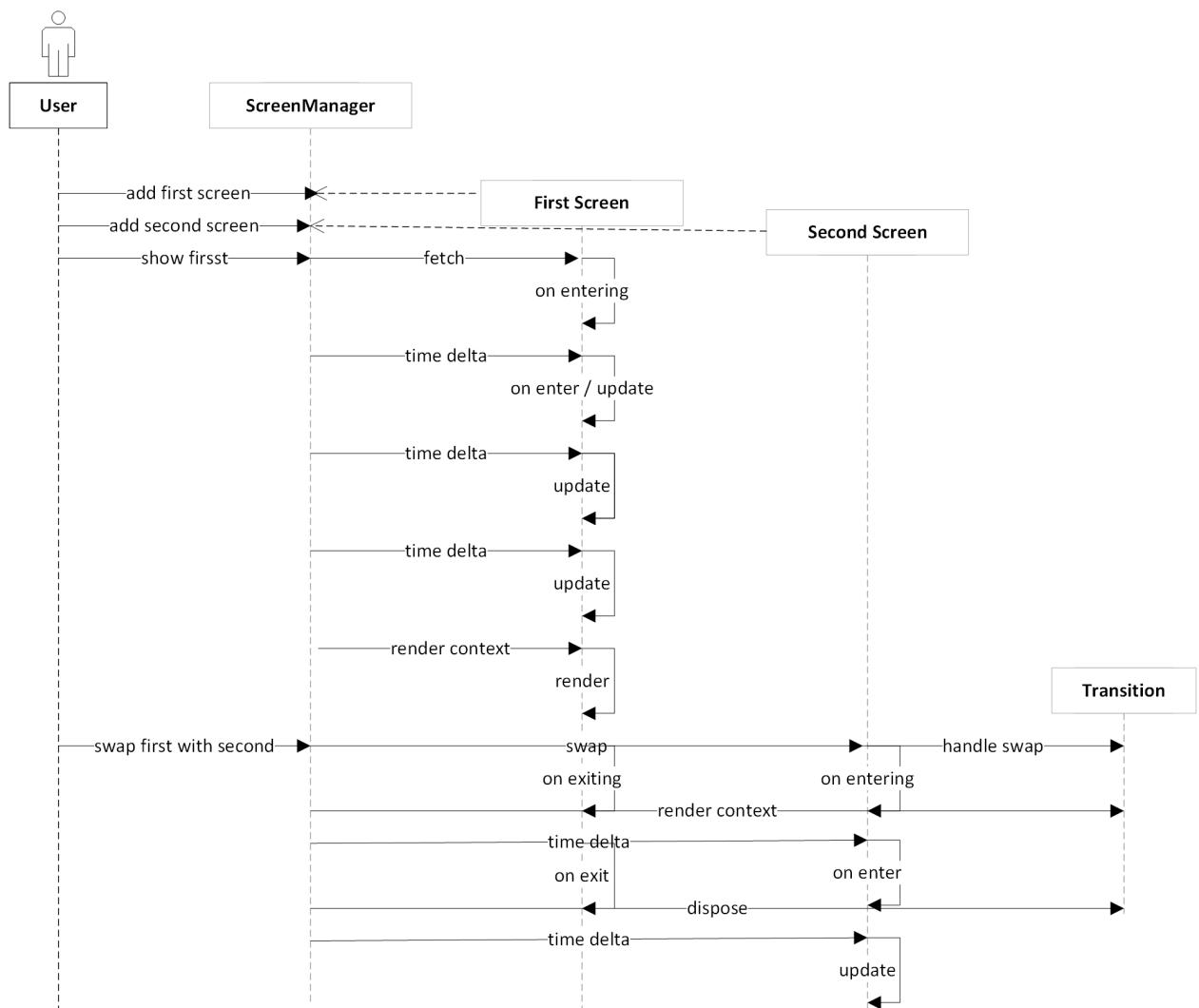
**Call-back driven** Τα υποσυστήματα παρέχουν call-back λειτουργικότητα, δηλαδή τη δυνατότητα δήλωσης κώδικας ο οποίος θα εκτελεστεί κατά κάποιο συμβάν. Ένα συμβάν μπορεί να είναι η σύγκρουση μεταξύ δύο αντικειμένων ή η είσοδος κάποιου παίχτη στον κόσμο. Ο χρήστης μπορεί για παράδειγμα να δηλώσει ότι 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 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 [12].

### 3.2.1 Απαιτήσεις συστήματος διαχείρισης σκηνής

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



**Διάγραμμα 3.3:** Διάγραμμα ακολουθίας του υποσυστήματος διαχείρισης οθονών

### 3.2.2 Συστατικά του συστήματος

**Κεντρικό σύστημα διαχείρισης** Οι λειτουργίες του συστήματος διαχείρισης είναι οι παρακάτω:

- Απόδοση και ενημέρωσης 1-N σκηνών με δυναμικά εναλλασσόμενη σειρά στο πλαίσιο του κύκλου ζωής του πυρήνα.
- Προετοιμασία των οθονών και δυνατότητα εύρεσης χρησιμοποιώντας κλειδιά.
- Εύκολη προβολή, απόκρυψη, εναλλαγή οθονών χρησιμοποιώντας κλειδιά.
- Δυνατότητα εξατομίκευσης της απόδοσης κατά τις διάφορες μεταβάσεις των σκηνών.

**Οικοδεσπότης οθόνης** Ο οικοδεσπότης οθόνης είναι υπεύθυνος για τα παρακάτω:

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

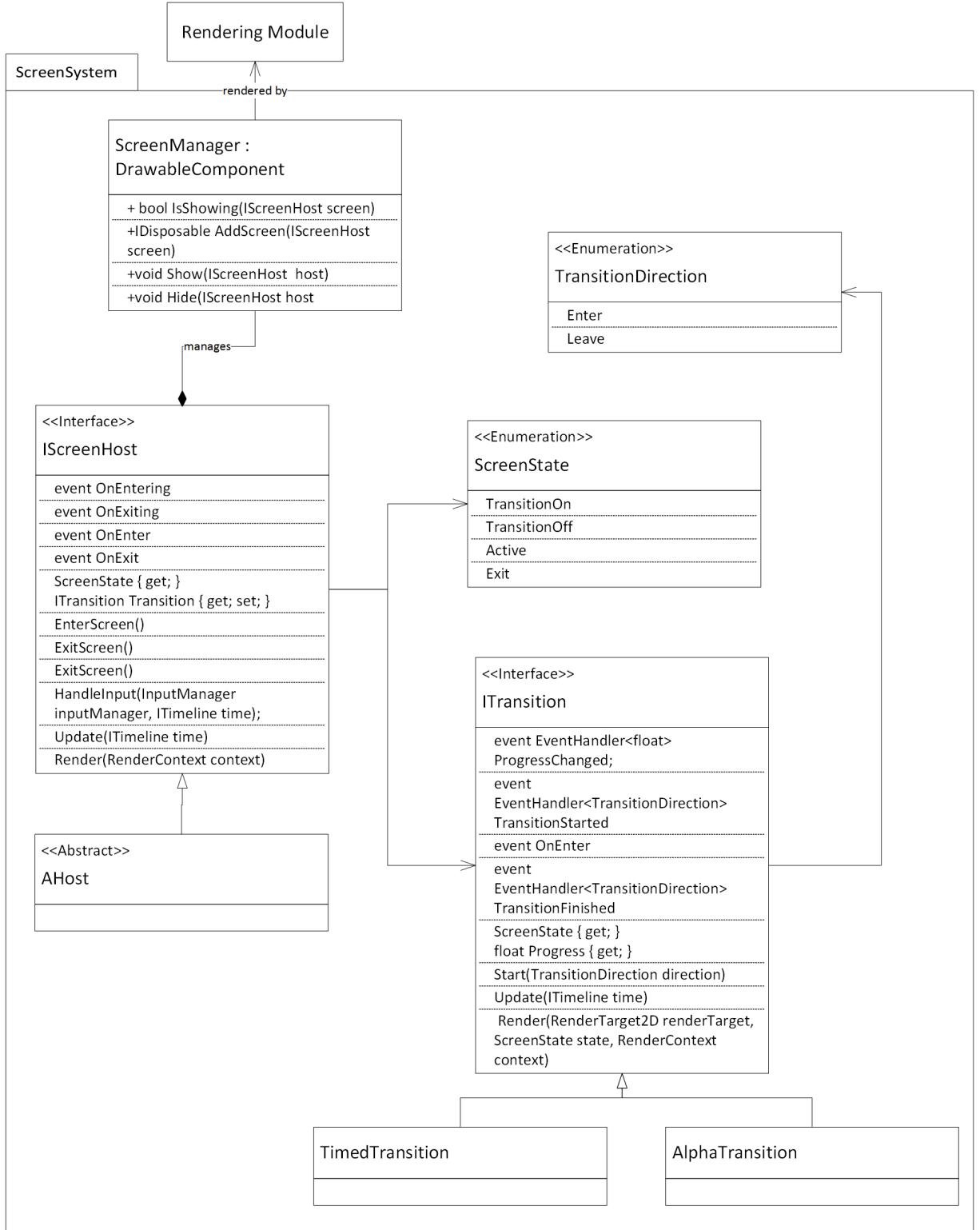
**Εναλλαγή οθονών** Η εξατομίκευση και παραμετροποίηση της εναλλαγής οθονών λαμβάνει μέρος μέσα στο γενικό πλαίσιο απόδοσης οθονών. Ο εξατομικευτής αναλαμβάνει την απόδοση του screen buffer στον οποίο αποδόθηκε η οθόνη, την κατάσταση στην οποία βρίσκεται η οθόνη, την κατεύθυνση και το ποσοστό εξέλιξης της εναλλαγής. Ο εξατομικευτής περιγράφεται στο παράδειγμα κώδικα 3.1.

#### Παράδειγμα κώδικα 3.1: Εξατομικευτής εναλλαγής οθονών

```
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:** UML του υποσυστήματος διαχείρισης οθονών

### 3.2.4 Παραδείγματα χρήσης API

Στο παρακάτω παράδειγμα γίνεται καταχώριση μιας οθόνης-σκηνής στο υποσύστημα με κάποιο κλειδί. Στη συνέχεια το υποσύστημα εμφανίζει την σκηνή με εξατομικευμένη διαδικασία εναλλαγής.

**Παράδειγμα κώδικα 3.2:** Παράδειγμα API του υποσυστήματος διαχείρισης οθονών-σκηνών

```

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 Συσκευές ανθρώπινης διεπαφής

Η μηχανή πρέπει να είναι σε θέση να διαβάζει, να επεξεργάζεται και να χρησιμοποιεί συσκευές ανθρώπινης διεπαφής (human interface devices) [8]. Η διαδικασία

ανάγνωσης χωρίζεται στις παρακάτω κατηγορίες:

**Polling** Η κατάσταση κάποιου συσκευών (κυρίως της παλιάς σχολής) διαβάζεται ρωτώντας τη συσκευή για την κατάστασή της περιοδικά. Αυτό καταλήγει σε πλεονασμό, γιατί ρωτάει πολλές φορές χωρίς να παίρνει απάντηση, και πιθανών να συμβάλει σε μια 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 καταχωρίσει κώδικα σε συμβάντα ώστε να εκτελείται μόνο όταν συμβεί κάποια αλλαγή κατάστασης.

### 3.3.2 Τύποι εισόδων

Οι διάφοροι τύποι εισόδων και οι πιθανές καταστάσεις τους είναι οι παρακάτω:

**Ψηφιακά κουμπιά** Τα ψηφιακά κουμπιά (digital buttons) έχουν δύο καταστάσεις: pressed / not pressed

**Αναλογικοί άξονες και κουμπιά** Οι αναλογικοί άξονες και κουμπιά (analog axes and buttons) επιστρέφουν εύρος τιμών: το βαθμό της πίεσης της σκανδάλης ή τη θέση του μοχλού στο δισδιάστατο άξονα.

**Αναφορικοί άξονες** Οι αναφορικοί άξονες (relative axes) 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.

**Επιταχυντές** Οι επιταχυντές (accelerators) ανιχνεύουν τρισδιάστατες επιταχύνσεις.

**Sensor bars** Αισθητήρες όπως οι κάμερες.

**Αφή και χειρονομίες** Η αφή και χειρονομίες (touch and gestures) είναι τύποι εισόδου σε οθόνες αφής όπως στις οθόνες στα έξυπνα τηλέφωνα.

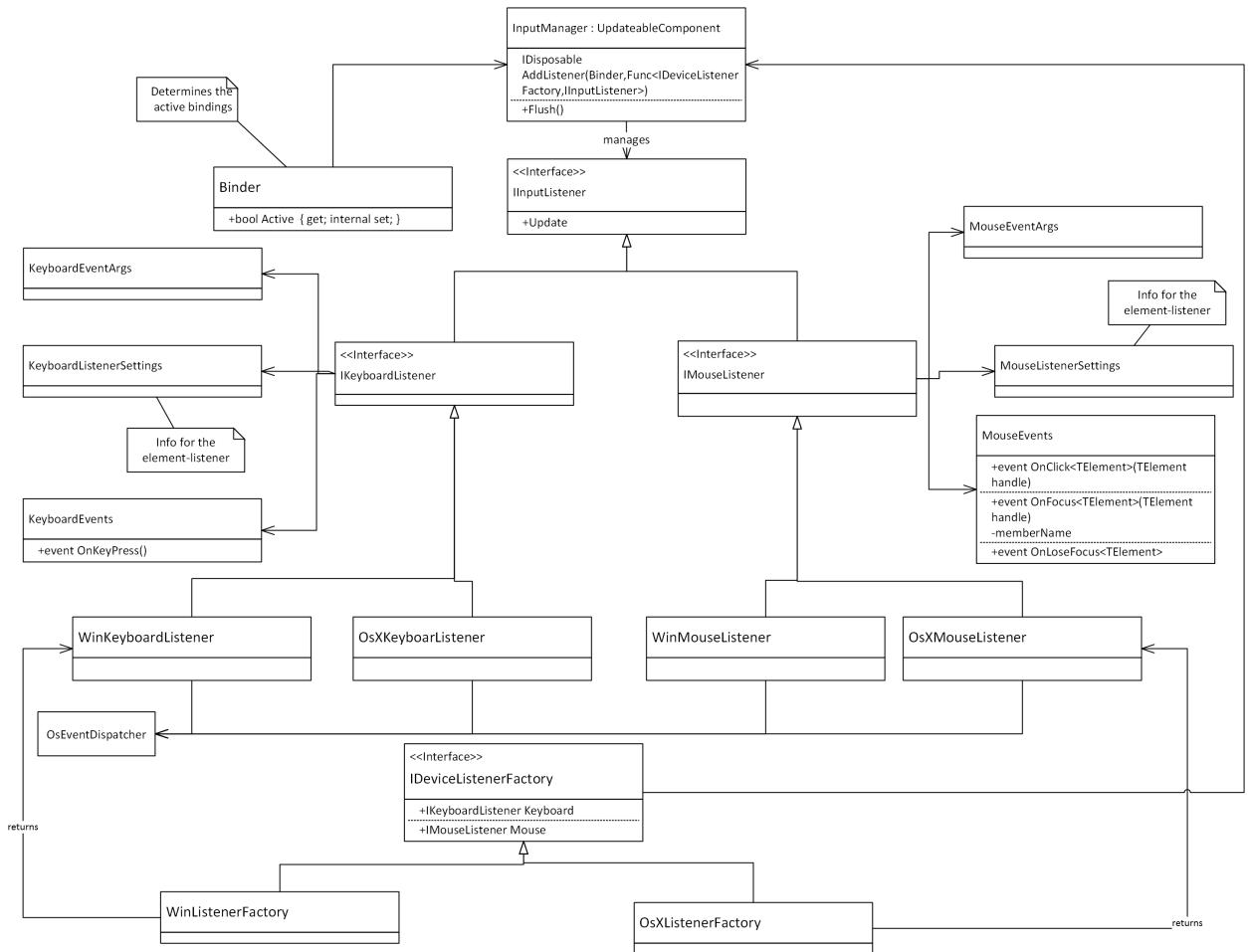
### 3.3.3 Απαιτήσεις του υποσυστήματος διαχείρισης εισόδων

Η οντότητα θέλει να εκτελέσει κώδικα προσανατολισμένο κατά συμβάν εισόδου από συσκευή ανθρώπινης διεπαφής. Ο κώδικας αυτός εκτελείται αντίστοιχα για ποντίκι, πληκτρολόγιο και χειριστήριο. Ο χρήστης μπορεί να αλλάξει συσκευή διεπαφής

ενώ βρίσκεται σε τρέχον παιχνίδι. Η βιβλιοθήκη δεν πρέπει να στηρίζεται σε κανένα υλικό ή συσκευή. Επίσης ο χρήστης χρειάζεται τη δυνατότητα να ακυρώνει συμβάντα.

### 3.3.4 Ακροατές

Το κάθε υποσύστημα διαχείρισης συσκευών ανθρώπινων διεπαφών αποτελείται από τον ακροατή (listener) για παράδειγμα τον MouseListener ή KeyboardListener, ο οποίος χρησιμοποιεί τις βιβλιοθήκες του τρέχον λειτουργικού για την επιστολή συμβάντων των διεπαφών. Ανάλογα με την πλατφόρμα στην οποία αναπτύσσεται το παιχνίδι, δημιουργούνται οι αντίστοιχοι ακροατές μέσω του abstract factory [6]. Ο χρήστης μπορεί να προσκολλήσει κώδικα ο οποίος να εκτελείται ανά συμβάν π.χ. στη μετακίνηση του ποντικιού. Οι ακροατές κατά την καταχώρηση συμβάντος, επιστρέφουν ένα αντικείμενο το οποίο μπορεί να χρησιμοποιηθεί για την κατάργησή του καταχωρημένου συμβάντος. Οι ακροατές ανάλογα με τις επιλογής 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 προσκολληθεί στην προετοιμασία του κύκλου ζωής σε υποσυστήματα διεπαφών. Επίσης με τη χρήση των active binders, μπορεί να γίνει απενεργοποίηση των ακροατών π.χ. όταν είναι ανοιχτό το menu επιλογών, οι ο ActiveBinder του παιχνιδιού είναι απενεργοποιημένος, με αποτέλεσμα τα συμβάντα τα οποία είναι προσκολλημένα στον διαχειριστή εισόδων να μην εκτελούνται. Στο UML διάγραμμα 3.5 παρουσιάζεται η αρχιτεκτονική του υποσυστήματος.



**Διάγραμμα 3.5:** Αρχιτεκτονική υποσυστήματος διαχείρισης εισόδων

**API** Στο παράδειγμα 3.3.4 παρουσιάζεται η δημιουργία και καταχώρηση των ακροατών.

### Παράδειγμα κώδικα 3.3: Παράδειγμα καταχώρησης ακροατών στο υποσύστημα εισόδων

```

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 εντοπισμός συγκρούσεων

Η φυσική στα παιχνίδια έχει ως βάση τα μαθηματικά. Η ανάλυση του συστήματος της φυσικής επικεντρώνεται σε μοντελοποίηση υψηλού επιπέδου. Η υλοποίησή του βασίζεται στην 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 πώς αλληλεπιδρούν τα διάφορα αντικείμενα και χαρακτήρες μέσα στον κόσμο. Όλα τα αλληλεπιδραστικά παιχνίδια είναι temporal simulations δηλαδή 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 είδη προθεσμιών (deadlines). Για να θεωρείται η προσομοίωση φυσικής σταθερή, πρέπει να ενημερώνεται τουλάχιστον 120 φορές το δευτερόλεπτο, ο μηχανισμός τεχνητής νοημοσύνης θα πρέπει να καλείται τουλάχιστον κάθε δευτερόλεπτο και οι buffers ήχου 60 φορές το δευτερόλεπτο για να αποτρέπονται δυσλειτουργίες του συστήματος διαχείρισης 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 διακριτά κατά τη διάρκεια της προσομοίωσης και είναι πιο συχνά γιατί η επόμενη κατάσταση της προσομοίωσης καθορίζεται από την εισαγωγή δεδομένων σε πραγματικό χρόνο από το χρήστη. [3]

### 3.4.2 Βασικές έννοιες φυσικής του συστήματος

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

**Μάζα** Η μάζα (mass) 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.

**Πυκνότητα** Η πυκνότητα (density) εκφράζει τη μάζα του υλικού που περιέχεται σε μία μονάδα όγκου.

**Force** Σε ότι αφορά τα ελεύθερα σώματα, η δύναμη (force) 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.

**Ροπή** Ροπή (torque) 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.

**Ωθηση** Η ώθηση (impulse) είναι το 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** Το law of restitution δηλαδή ανάκτηση με βάση το κέρδος. Η υποχρέωση της οντότητας να αποζημιώσει από τα κέρδη τη για κάποιο γεγονός.

**Απόσβεση** Η απόσβεση (damping), είναι η επιρροή εντός η κατά ενός συστήματος ταλάντωσης που έχει ως αποτέλεσμα τη μείωση, τον περιορισμό η την πρόληψη των ταλαντώσεων του. Σε φυσικά συστήματα , απόσβεση παράγεται με διαδικασίες που διαχέουν την ενέργεια που αποθηκεύεται στο ταλάντωση.

### 3.4.3 Οντότητες του συστήματος

Οι οντότητες που απαρτίζουν το υποσύστημα φυσικής είναι οι παρακάτω:

**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** Το 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** Το body περιέχει της πληροφορίες ενός αντικειμένου του οποίου δεν βλέπεις ούτε συγκρούεσαι και έχει τις παρακάτω ιδιότητες:

**Mass** Το βάρος

**Velocity** η ταχύτητα και η κατεύθυνση της κίνησης υπό μορφή διανύσματος.

**Rotation inertia** πόσος κόπος χρειάζεται για να ξεκινήσει η περιστροφή ή η κίνηση

**Angular velocity** πόσο γρήγορα και σε ποια κατεύθυνση περιστρέφεται

**Position** 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 προσομοίωση του body.

**Joint** Το joint συνδέει δύο ή περισσότερα bodies μεταξύ τους.

**Joint Motor** Ο οδηγός της κίνησης των συνδεδεμένων με joint bodies.

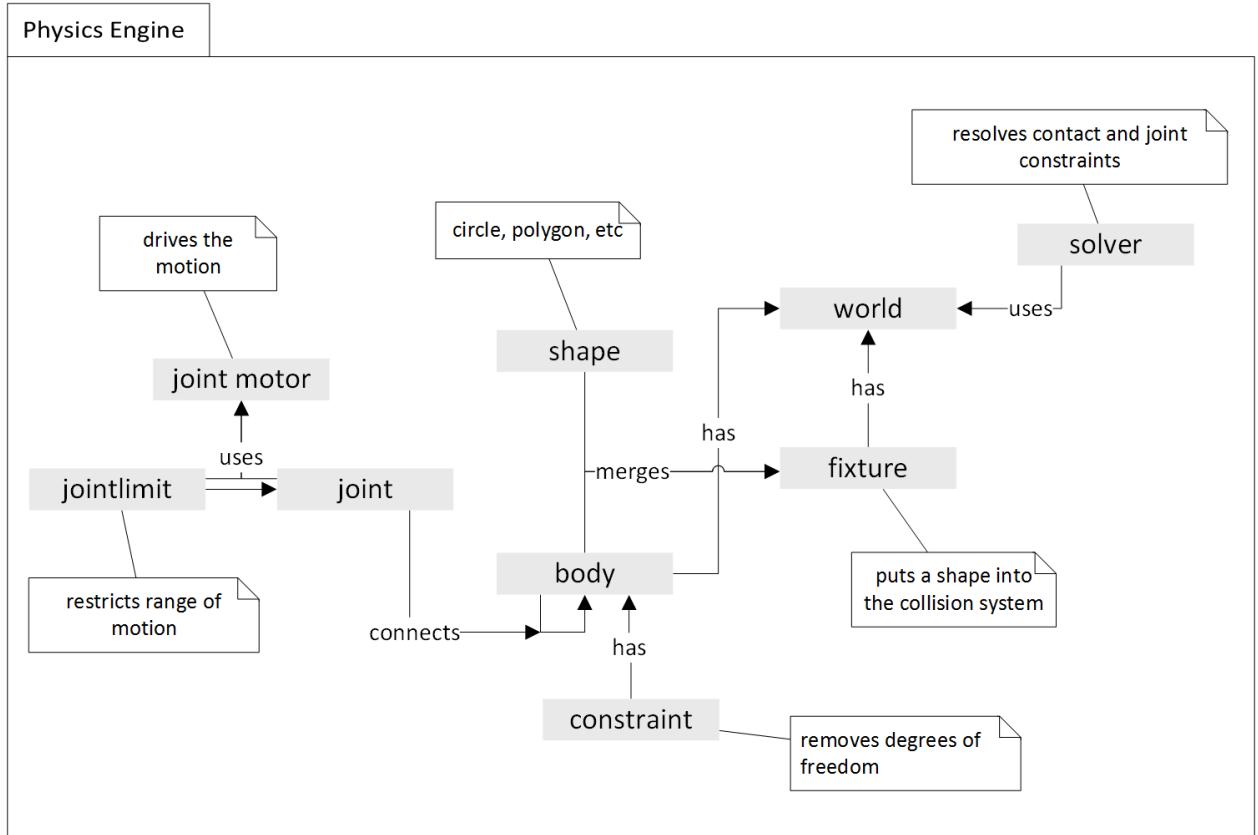
**Joint Limit** Το joint limit περιορίζει το εύρος κίνησης του joint motor.

#### 3.4.4 Αρχιτεκτονική

Οι σχέσεις μεταξύ των οντοτήτων παρουσιάζονται στο διάγραμμα 3.6

### 3.5 Διεπαφή χρήστη

Η επικοινωνία του χρήστη με τη μηχανή γίνεται συνήθως με συσκευές ανθρώπινης διεπαφής. Η επικοινωνία αυτή αντί να περιορίζεται σε ένα απλό σήμα εισόδου (input



**Διάγραμμα 3.6:** Υποσύστημα φυσικής και εντοπισμού συγκρούσεων

signal), μπορεί να γίνει εύκολα, αυτονόητα, αποτελεσματικά και φιλικά προς το χρήστη. Διεπαφή χρήστη (user interface) 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 ενδείξεις και εργαλεία μέσω γραφικών, προκειμένου ο χρήστης να φέρει εις πέρας κάποιες επιθυμητές λειτουργίες.

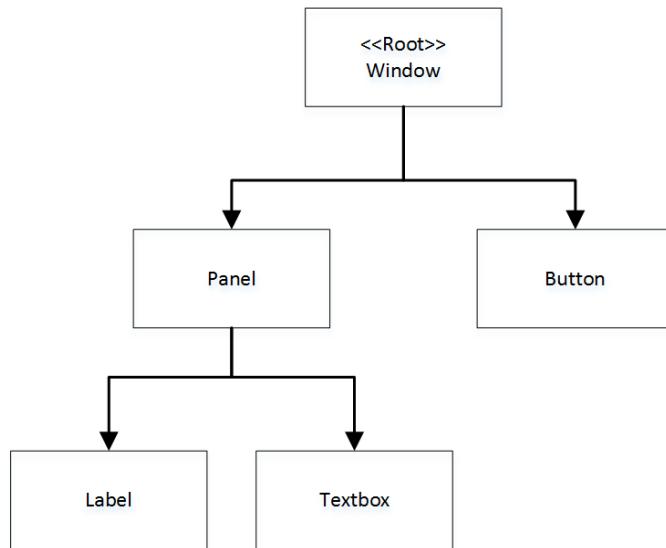
### 3.5.1 Απαιτήσεις

Κατά το σχεδιασμό ενός παιχνιδιού συχνά δημιουργείται η ανάγκη για επικοινωνία με τον παίχτη μέσω διεπαφή χρήστη. Η δομή του, το στυλ της απόδοσης (χρώμα, textures, fonts κλπ), η απόδοση (rendering), η συμπεριφορά και ο εκτελέστιμος κώδικας κατά τα συμβάντα της διεπαφής πρέπει να είναι αποσυνδεδεμένα έτσι ώστε να μην επηρεάζει το ένα το άλλο. Επίσης η διεπαφή χρήστη προορίζεται για διάφορες αναλύσεις οθόνης 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 (browser) να φορτώσει μια ιστοσελίδα στη μνήμη από κάποιον web server, ο περιηγητής αναλύει το html αρχείο και οργανώνει τα στοιχεία στη μνήμη σε δομή δέντρου B-Tree. Λόγω της οργάνωσης σε δέντρο, δημιουργούνται σχέσεις μεταξύ των στοιχείων.



**Διάγραμμα 3.7:** B-Tree διεπαφής χρήστη

Στο παράδειγμα 3.7 το window είναι η ρίζα (root) του δέντρου. Το window έχει δύο παιδιά, ένα panel και ένα button. Το button έχει και αυτό δύο παιδιά: ένα textbox και ένα label. Τα button, textbox και label είναι leaves (φύλλα) του δέντρου, το panel είναι κόμβος (node), και το panel και το button αδέλφια (siblings).

**Πλεονεκτήματα της οργάνωσης σε 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 στοιχηθούν και να αποδοθούν ανάλογα με τη σχέση τους με συγγενικά στοιχεία. Καθώς διαβαίνεται (traversing) το δέντρο, το κάθε στοιχείο αποδίδεται σε πιο πάνω επίπεδο μέσα στο πλαίσιο του πατέρα, δηλαδή το στοι-

χείο παιδί βρίσκεται ”πάνω” από τον πατέρα, ούτως ώστε να δημιουργείται η ψευδαίσθηση του βάθους.

**Καλύτερη απόδοση στην οθόνη** Όταν κάποιος κόμβος δεν διατέμνεται με το πλαίσιο του παράθυρου, δεν αποδίδεται στην οθόνη και η διάβαση του δέντρου σταματά.

**Αποσαφήνιση σειράς συμβάντων** Κατά τα συμβάντα στη διεπαφή χρήστη, 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 λογαριθμικό χρόνο.

Στο UML διάγραμμα 3.8 παρουσιάζεται το υποσύστημα το οποίο είναι υπεύθυνο για το κτίσιμο της δομής της διεπαφής χρήστη στη μνήμη.

### 3.5.3 Τα υποσυστήματα διεπαφής χρήστη

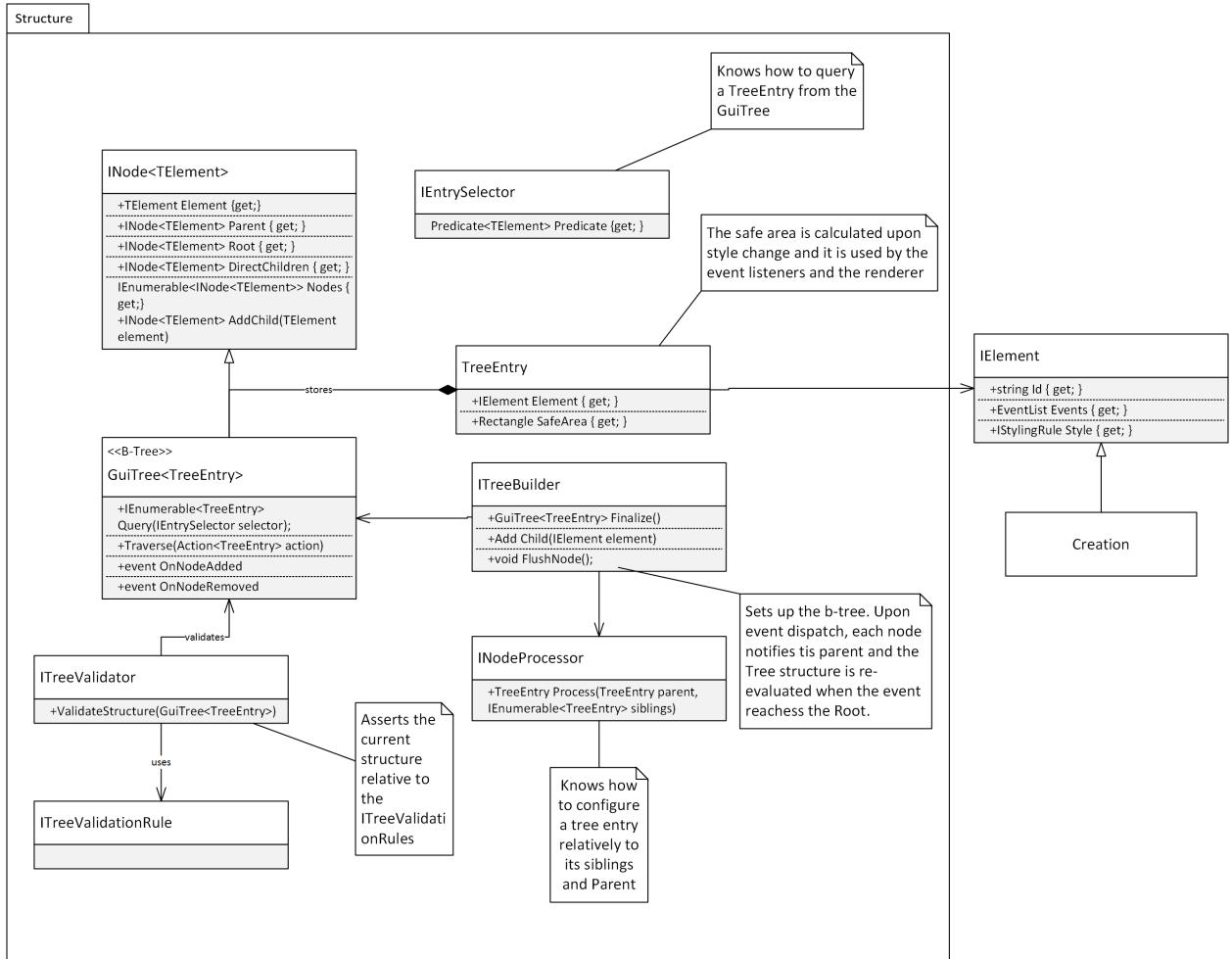
Το σύστημα διεπαφής χρήστη χωρίζεται στα παρακάτω υποσυστήματα

**Οργάνωσης δομής** Το υποσύστημα οργάνωσης δομής αναλαμβάνει το κτίσιμο και την αποθήκευση της ιεραρχίας των στοιχείων στη μνήμη.

**Style definition** Το style definition οποίο εμπλουτίζει την απόδοση με animations, textures κλπ.

**Factory** Μέσω του factory ο οποίο ο χρήστης της μηχανής δημιουργεί στοιχεία. Το υποσύστημα αυτό χρησιμοποιεί το abstract factory pattern [6] για δημιουργία στοιχείων ανεξαρτήτου πλατφόρμας, και fluent builder για τη δημιουργία στοιχείων με φιλικό στον χρήστη API.

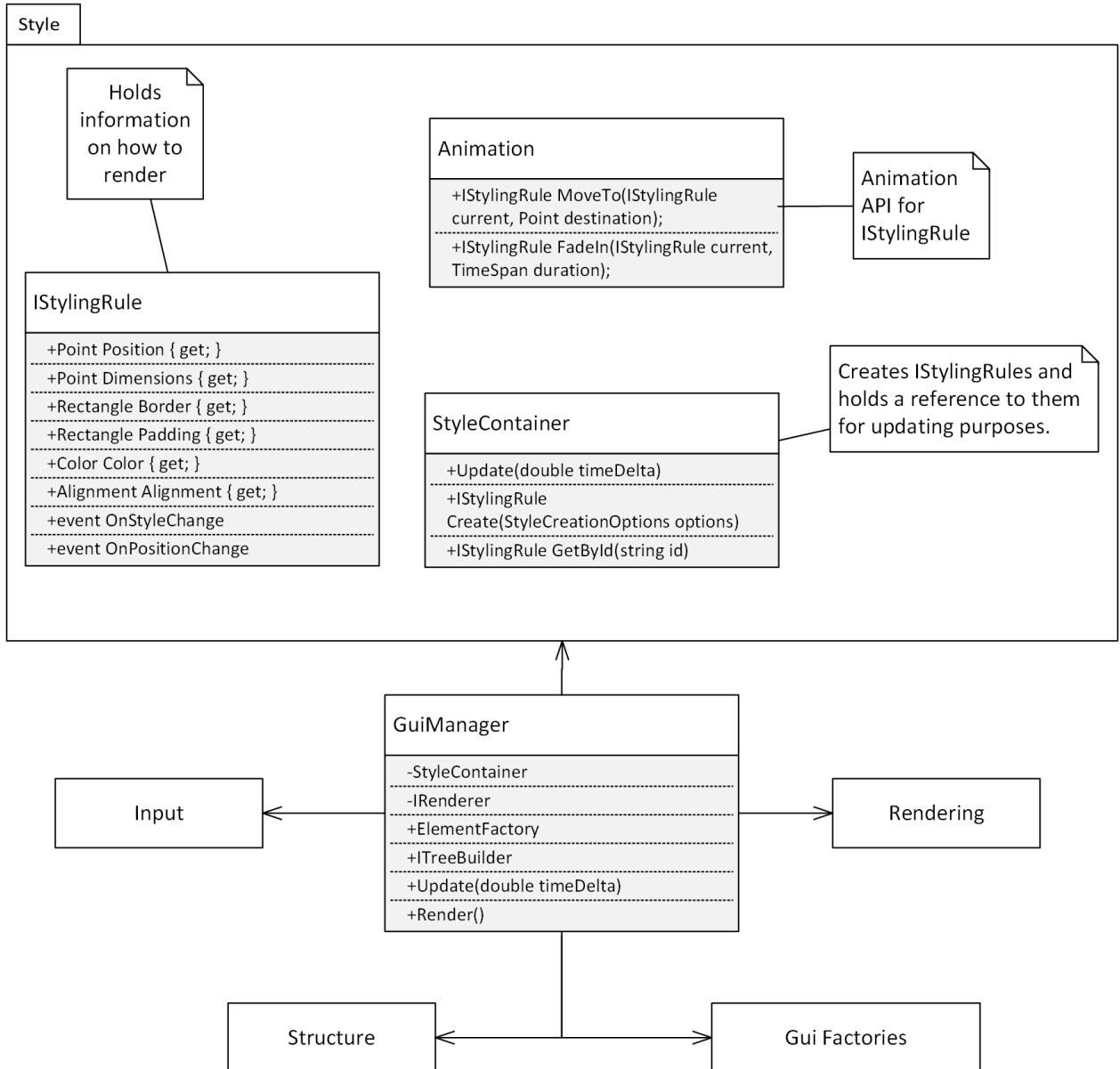
Οι σχέσεις μεταξύ των υποσυστημάτων του συστήματος διεπαφής χρήστη παρουσιάζονται στο UML διάγραμμα 3.9.



Διάγραμμα 3.8: Δομή διεπαφής χρήστη

### 3.5.4 Κύκλος ζωής διεπαφής χρήστη

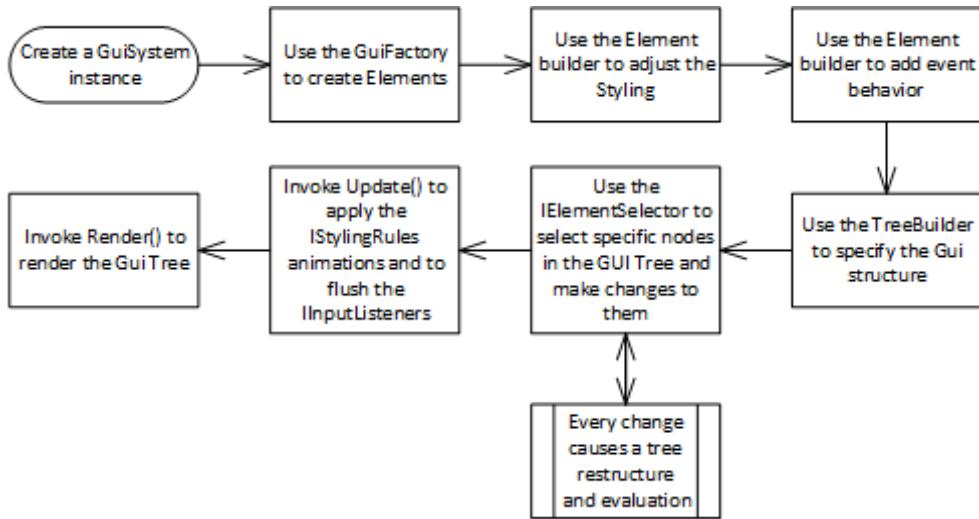
Η σειρά προετοιμασίας διαμόρφωσης και παρουσίασης της διεπαφής χρήστη φαίνεται στο διάγραμμα 3.10.



**Διάγραμμα 3.9:** UML του συστήματος ανθρώπινης διεπαφής

### 3.6 Διαχείριση πόρων

Τα παιχνίδια απαρτίζονται από πολλά είδη δεδομένων, είτε αυτά είναι τρισδιάστατα μοντέλα, textures, 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 διαχειριστής πόρων (asset / resource manager), ο οποίος χειρίζεται το σύστημα αρχείων του λειτουργικού. Ο διαχειριστής πόρων 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



**Διάγραμμα 3.10:** Κύκλος ζωής διεπαφής χρήστη

δεδομένων, για να μην μπλοκάρεται η ροή του παιχνιδιού.

Ο διαχειριστής πόρων απαρτίζεται από δύο μέρη. Το πρώτο μέρος, ο offline resource manager, περιλαμβάνει εργαλεία τα οποία χειρίζονται assets και τα μεταφράζουν σε μια μορφή η οποία είναι επεξεργάσιμη από τη μηχανή. Το δεύτερο μέρος, ο runtime resource manager διαχειρίζεται τα assets στη μνήμη, φροντίζει να είναι διαθέσιμα όταν χρειάζονται και να αφαιρεθούν από τη μνήμη όταν δεν θα ξαναχρησιμοποιηθούν. Τα περισσότερα assets δεν φορτώνονται στη μνήμη με την κανονική τους μορφή (format). Τα assets περνούν από κάποιο conditioning pipeline για να μετασχηματιστούν σε μια μορφή η οποία μπορεί να επεξεργαστεί η μηχανή. Ένα τρισδιάστατο μοντέλο δεν περιέχει επαρκή πληροφορία και η μηχανή δεν ξέρει για τι προορίζεται. Ο resource manager εμπλουτίζει τα assets με μεταδεδομένα (metadata) το οποία περιγράφουν το πώς η μηχανή θα χειρίστει το κάθε asset. Η βάση δεδομένων πόρων (resource database) ξέρει πώς να χειρίζεται διάφορα είδη assets σε μια μορφή την οποία η μηχανή καταλαβαίνει. Μία τεχνική είναι 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 εκδόσεις (versioning) για να εντοπίζονται οι αλλαγές κατά τη πορεία ανάπτυξης. Η βάση μπορεί να είναι είτε SQL είτε απλά ένα B-Tree με buffer blocks και δυνατότητα προσπέλασής τους [9].

**Στάδια του αγωγού πόρων** Τα στάδια του αγωγού πόρων (resource pipeline) είναι τα παρακάτω:

**Granular resources** Πόροι οι οποία συνδέονται με τις οντότητες στο παιχνίδι.

**Σύνδεση με κώδικα** Ευκολία στη σύνδεση των δεδομένων με τον πηγαίο κώδικα

**Εξαγωγή** Εξαγωγή δεδομένων σε διάφορες μορφές.

**Συμβατότητα** Συμβατότητα με άλλες μορφές ή δυνατότητα μετατροπής τους σε μορφή η οποία είναι συμβατή με τη μηχανή.

**Εύκολο κτίσιμο** Ευκολία κατά το κτίσημο, αφού η μηχανή αναλαμβάνει τις εξαρτήσεις μεταξύ δεδομένων και κώδικα.

### 3.6.1 Ευθύνες του offline resource manager

**Exporters** Η δυνατότητα μετατροπής από native μορφή σε μορφή η οποίο μπορεί να τροποποιηθεί από τη μηχανή.

**Resource Compilers** Κατά την εξαγωγή χρειάζεται να γίνει κάποιου είδους καμουφλάρισμα των δεδομένων ώστε να συμβαδίζουν με την μηχανή.

**Resource Linkers** Πολλές φορές περισσότερα από ένα αρχεία χρειάζονται να συνδέθούν 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 μοναδικοί πόροι και περισσότεροι από ένα του ίδιου τύπου.
- Διαχειρίζεται το πόσο χρόνο είναι διαθέσιμα στη μνήμη
- Φορτώνει και ξεφορτώνει από τη μνήμη.
- Χειρίζεται composite resources δηλαδή πόρους οι οποίοι αποτελούνται από περισσότερους από ένα. Ένα τρισδιάστατο μοντέλο για παράδειγμα, αποτελείται από 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.
- Επιτρέπει την τροποποίηση των δεδομένων αφού φορτωθούν στη μνήμη
- Προσφέρει τη δυνατότητα ασύγχρονης φόρτωσης για παραλληλισμό ενεργειών.

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

**Τεχνικές διαχείρισης πόρων στη μνήμη** Μια συνηθισμένη τεχνική είναι το flyweight pattern [6], 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. Όταν το παιχνίδι χρειάζεται κάποιο resource, η μηχανή ελέγχει αν υπάρχει το κλειδί. Αν υπάρχει τό επιστρέφει και αν όχι τότε φορτώνει στο hashtable.

### Στάδια του κύκλου ζωής των πόρων στη μνήμη

**Καθολικοί πόροι** Οι καθολικοί πόροι (global resources) φορτώνονται στην αρχή του παιχνιδιού και χρειάζονται συνέχεια.

**Πόροι επιπέδου** Οι πόροι αυτοί χρειάζονται στην έκταση συγκεκριμένου επιπέδου.

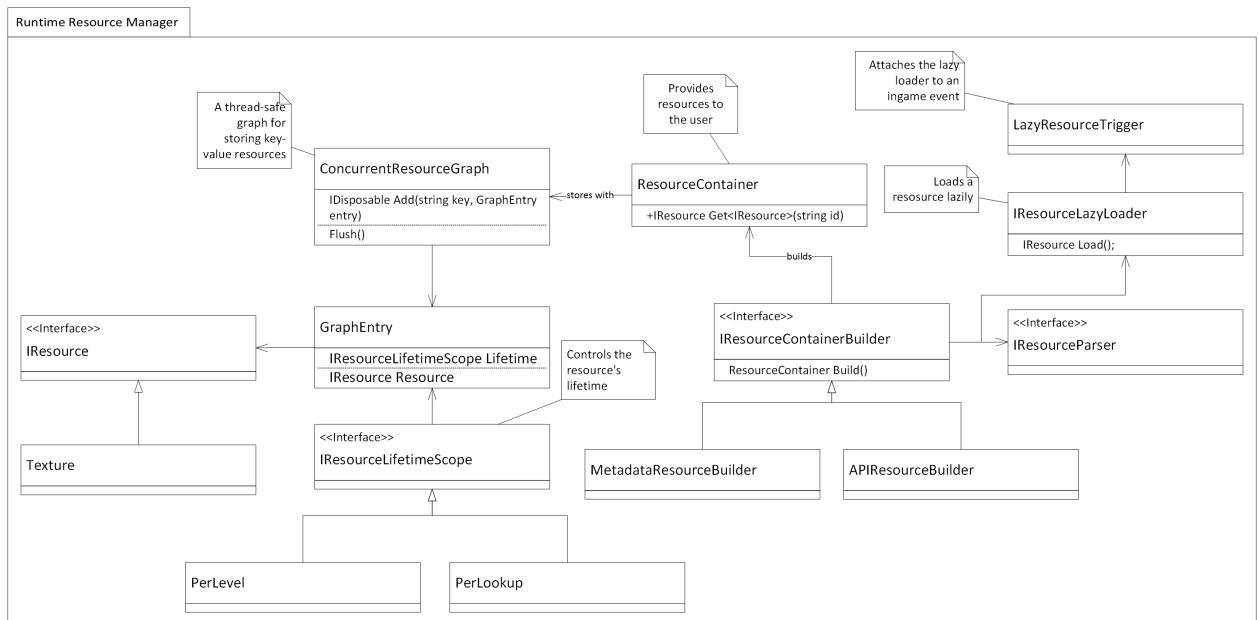
**Σύντομης διάρκειας ζωής** Οι πόροι σύντομης διάρκειας ζωής (short-living resources) χρειάζονται για ένα συγκεκριμένο αυτοτελή σκοπό π.χ. μια σκηνή η οποία εμφανίζεται μια φορά κατα τη διάρκεια ενός επιπέδου.

**Ζωντανής ροής** Οι πόροι ζωντανής ροής (live streamed resources) είναι για παράδειγμα αρχεία μουσικής και ηχητικών εφέ τα οποία διαβάζονται από το δίσκο δυναμικά και φορτώνονται στη μνήμη σε κομμάτια (chunks).

**Τεχνικές διαχείρισης κύκλου ζωής** Υπάρχουν διάφορες τεχνικές για τη διαχείριση του κύκλου ζωής των πόρων. Μια καλή τεχνική είναι να φυλάγονται οι αναφορές (references) στα αντικείμενα κάθε φορά που γίνεται προσπέλαση στη μνήμη π.χ. στην αρχή κάθε επιπέδου. Μετράται πόσες φορές ζητήθηκε ο συγκεκριμένος πόρος.

1. Βρίσκουμε όλα τα resources που χρειάζονται για τη συγκεκριμένη σκηνή και ανξάνουμε τον μετρητή τους κατά 1. Αφαιρούμε στα υπόλοιπα 1.
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: UML του διαχειριστή πόρων πραγματικού χρόνου

### 3.7 Τεχνητή νοημοσύνη

Η τεχνητή νοημοσύνη χρησιμοποιείται για να παράγει ενφυείς 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 εντελώς 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 [9].

### 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 [2].

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 [6].

**Inverter** Ο κόμβος ο οποίος είναι decorated από το inverter, επιστρέφει το αντίστροφο αποτέλεσμα κατά την αξιολόγηση.

**Repeat Until Success** Ο κόμβος αξιολογείται ως Running εφόσον αξιολογηθεί με αποτυχία.

**Repeater** Η αξιολόγηση του κόμβου επαναλαμβάνεται ανάλογα με το προκαθορισμένο αριθμό επαναλήψεων

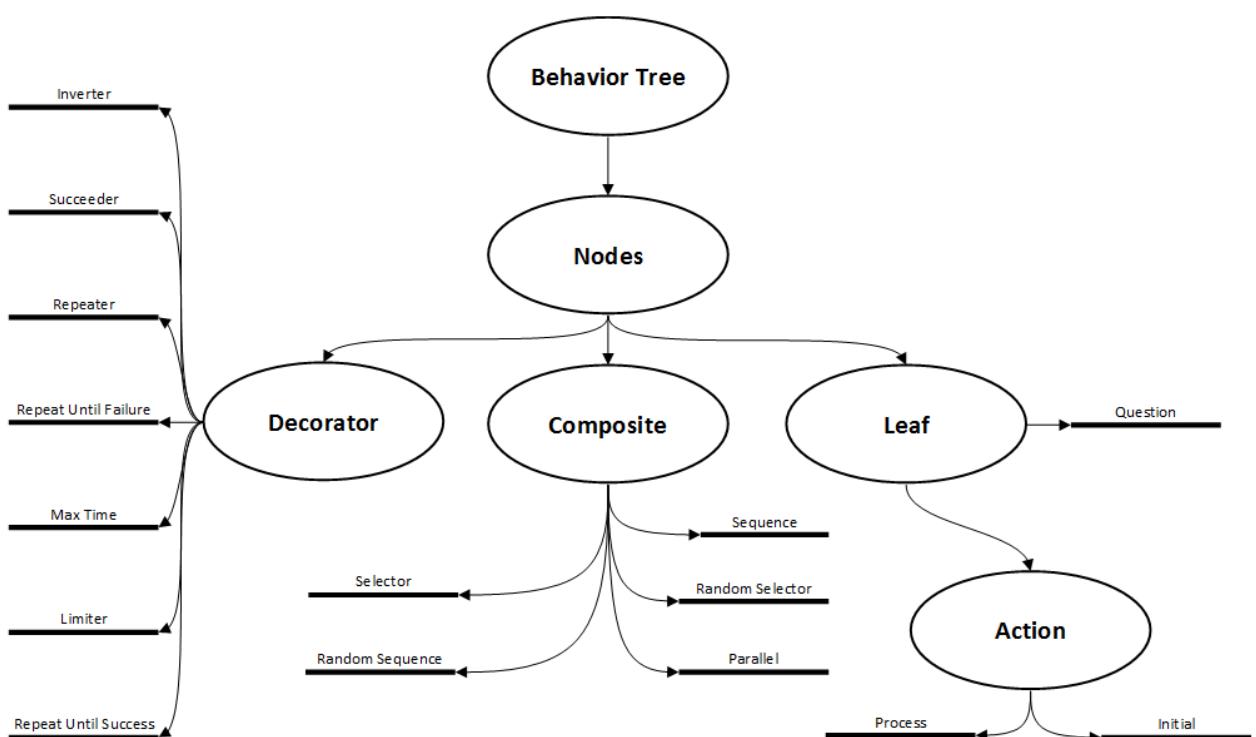
**Max Time** Η αξιολόγηση γίνεται μέσα σε στα πλαίσια χρονικού διαστήματος.

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

**Ερωτήσεις** Για παράδειγμα: υπάρχει το συγκεκριμένο αντικείμενο στο χάρτη;

**Συμπεριφορά** Για παράδειγμα: προχώρα ένα βήμα μπροστά.

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



Διάγραμμα 3.12: Δέντρο συμπεριφορών

**API** Στο API του υποσυστήματος συμπεριλαμβάνεται ένα fluent tree builder για εύκολη δημιουργία δέντρων στη μνήμη με κώδικα και ένας οραματιστής δέντρου (tree visualizer) για απόδοση του δέντρου στην οθόνη με αξιολόγηση και ενημέρωση πραγματικού χρόνου για αποσφαλμάτωση. Στο παρακάτω παράδειγμα θα γίνει μοντελοποίηση της συμπεριφοράς ενός χαρακτήρα ο οποίος προσπαθεί να βρει την έξοδο σε ένα λαβύρινθο με εχθρούς.

### Παράδειγμα κώδικα 3.4: Behavior tree builder

```

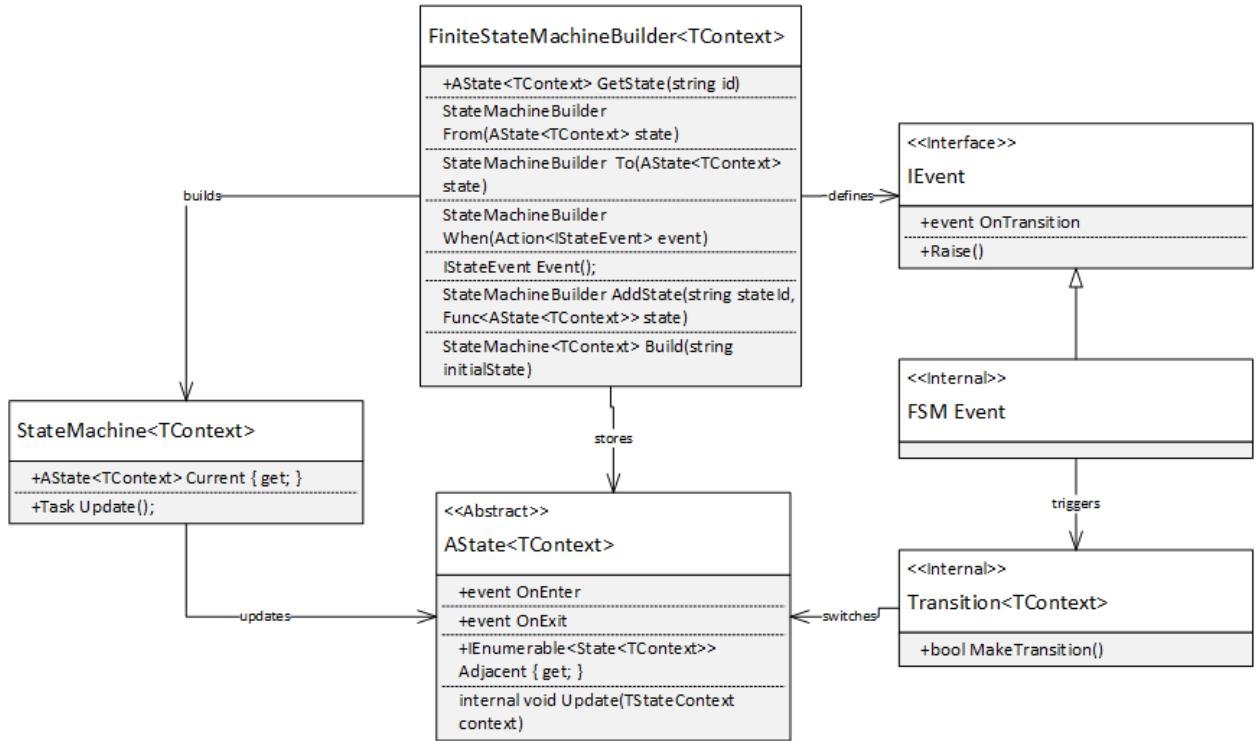
1 behaviorBuilder
2   . Selector(name: "MoveToEnd")
3     . Decorate(For.RepeatUntilFailure)
4       . Sequence(name: "tryWalk")
5         . Question(CheckNextTile, name: "next_
tileEmpty?")
6         . Behavior(MoveOneTile, name: "go1_
stepForward")
7       . Sequence("keyObstacle")
8         . Question(CheckTileForKey, name: "_
foundKey?")
9         . Selector("handleKey")
10      . End
11      . Sequence
12        . DecorateFor(Decorator.AlwaysSuccess)
13          . Behavior(RunAway)
14        . End
15      . End
16    . Tree;

```

### 3.7.2 Finite State Machines

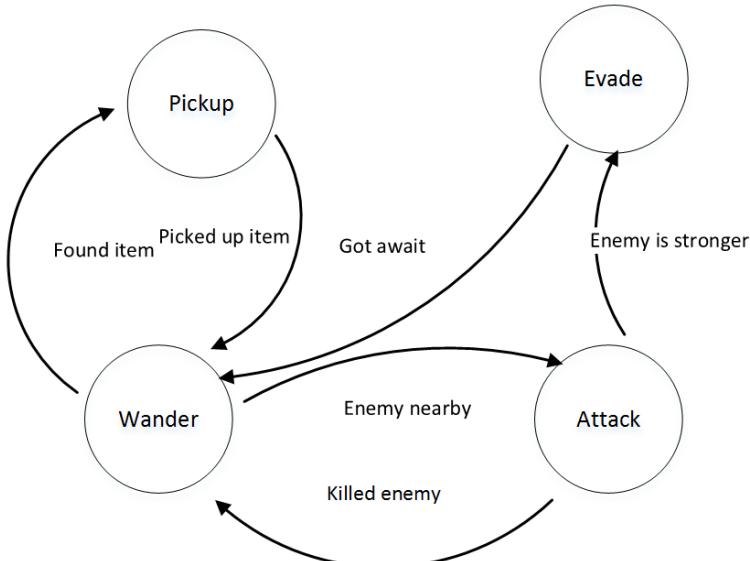
Finite state machine είναι μια αφηρημένη μηχανή η οποία μπορεί να βρίσκεται σε μία από ένα πεπερασμένο αριθμό καταστάσεων και με την επέλευση κάποιου γεγονότος μπορεί να αλλάξει από μια κατάσταση σε άλλη. Κατάσταση (state) είναι η περιγραφή της κατάστασης στην οποία βρίσκεται το σύστημα [7]. Γεγονός (event) δράση η οποία οδηγεί σε αλλαγή κατάστασης. Transition (μετάβαση) είναι ένα σύνολο δράσεων 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.

Στο παράδειγμα 3.14 μοντελοποιούνται οι καταστάσεις και τα γεγονότα τα οποία οδηγούν σε αλλαγή κατάστασης ενός χαρακτήρα με νοημοσύνη. Ο χαρακτήρας αυτός



Διάγραμμα 3.13: UML του υποσυστήματος των state machines

περιπλανιέται τυχαία, όταν συναντήσει αντικείμενο το μαζεύει και όταν συναντήσει εχθρό τον πολεμά. Αν ο εχθρός είναι πιο δυνατός, τότε αποφεύγει την μάχη φεύγοντας.



Διάγραμμα 3.14: Παράδειγμα state machine

Η σύνδεση του finite state machine γίνεται μέσω του builder [6]. Στον builder γί-

νεται καταχώρηση των καταστάσεων μέσω μιας μεθόδου-factory και δημιουργία αυτο-ενεργοποιημένων συμβάντων ή συμβάντων τα οποία ενεργοποιούνται από τον χρήστη τα οποία οδηγούν σε αλλαγή κατάστασης. Μετά γίνεται το κτίσιμο του FSM και η ενημέρωση του με βάση το γενικότερο πλαίσιο.

**Παράδειγμα κώδικα 3.5: State machine API**

```

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. 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 επιπλέον στοιχεία κοινωνικοποίησης όπως η επικοινωνία μέσω μηνυμάτων ή φωνής.

### 4.0.3 Αλληλεπίδραση σε εικονικό περιβάλλον

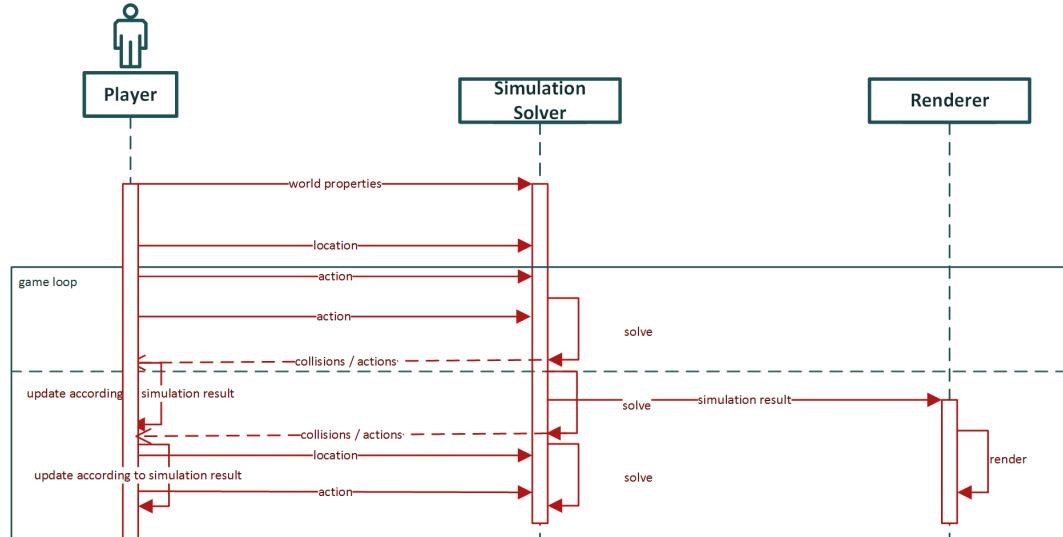
Ένας εικονικός κόσμος, περιλαμβάνει πολλές οντότητες οι οποίες αλληλεπιδρούν μεταξύ τους μέσω των μηχανισμών, νόμων και κανόνων που διέπουν τον κόσμο. Παράδειγμα μηχανισμού είναι η προσομοίωση του φυσικού κόσμου, όπου οι οντότητες ανταποκρίνονται σε νόμους της φυσικής.

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 [8].

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

#### 4.0.4 Οι απαιτήσεις του υποσυστήματος δικτύωσης

Με βάση τις απαιτήσεις του υποσυστήματος καταλήγουμε στο διάγραμμα ακολουθίας 4.1.



Διάγραμμα 4.1: Διάγραμμα ακολουθίας του υποσυστήματος δικτύωσης

Βλέποντας το διάγραμμα καταλαβαίνουμε ότι σε η διαδικασία απόδοσης περιλαμβάνει στατικά στοιχεία και αλγόριθμους, τα οποία 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.

## 4.1 Ανάλυση εννοιών

Η ανάγκη αποστολής πολλών μηνυμάτων ανά δευτερόλεπτο οδηγεί στη χρήση των sockets δικτύου (network sockets). Τα sockets χρησιμοποιούνται ένα IP και port και

επιτρέπουν την αποστολή και παραλαβή μηνυμάτων με βάση κάποιου πρωτοκόλλου.

#### 4.1.1 Πρωτόκολλο

Τα πιο συχνά χρησιμοποιούμενα πρωτόκολλα για δικτύωση σε παιχνίδια είναι τα παρακάτω [12]:

**TCP** Το 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** Το 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, η οποίο γίνεται 20 φορές ανά δευτερόλεπτο [11], 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.

#### 4.1.2 Αρχιτεκτονική δικτύου

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

**Client-server model** Στο client server model (μοντέλο πελάτη εξυπηρετητή) ο οποίο ο client απλά αποδίδει στην οθόνη και το μεγαλύτερο κομμάτι της λογικής και της προσομοίωσης τρέχει στον server. Ο server στέλνει οδηγίες στον client για το τι να αποδόσει 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** Στο peer to peer μοντέλο, 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 προσομοιώσεις 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 και συντήρηση γιατί δημιουργούνται race conditions στο χρονικό διάστημα αποστολής-παραλαβής μηνυμάτων που προορίζονται σε αλληλοεξαρτώμενες λειτουργίες.

## 4.2 Σχεδίαση του υποσυστήματος

Κατά το σχεδιασμό διαδικτυακών παιχνιδιών πολλά μοτίβα και στερεότυπα κώδικα επαναλαμβάνονται χωρίς να συμβάλλουν στην λογική ή μηχανισμούς. Σκοπός της βιβλιοθήκης 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.

### 4.2.1 Οντότητες

**Serialization** Σε μια αντικειμενοστραφή γλώσσα χρησιμοποιούνται κλάσεις για να μοντελοποιήσουν το πρόβλημα. Όμως τα αντικείμενα των κλάσεων δεν μπορούν να σταλούν μέσω socket δικτύου στην αρχική τους μορφή, πρέπει να γίνει μια μετατροπή σε binary για να είναι συνεπής με τη μορφή 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.

### 4.2.2 Τύποι μηνυμάτων

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

**Αλλαγή κατάστασης** Τα παρακάτω μηνύματα αποστέλλονται όταν συμβεί αλλαγή κατάστασης του παιχτη στο δίκτυο του παιχνιδιού.

**Connecting** Ο παιχτης ξεκίνησε τη διαδικασία σύνδεσης.

**Connected** Ο παιχτης συνδέθηκε.

**Disconnecting** Ο παιχτης ξεκίνησε τη διαδικασία αποσύνδεσης.

**Disconnected** Ο παιχτης αποσυνδέθηκε.

**Data** Το μήνυμα περιέχει δεδομένα.

**ErrorMessage** Το μήνυμα περιέχει πληροφορίες για σφάλμα.

**WarningMessage** Το μήνυμα περιέχει προειδοποίηση.

**VerboseMessage** Το μήνυμα περιέχει γενικές πληροφορίες για την κατάσταση του δικτύου.

**ConnectionApproval** Το μήνυμα σηματοδοτεί την έγκριση στο δίκτυο

**DiscoveryRequest** Σε περιπτώσεις ανιχνεύσης server σε δίκτυο, ο παιχτης στέλνει το αίτημα ανίχνευσης.

**DiscoveryResponse** Σε επιτυχείς ανιχνεύσεις ο server επιστρέφει απάντηση με πληροφορίες.

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

#### 4.2.3 Διαχειριστής δικτύωσης

Ο διαχειριστής δικτύωσης (network manager) είναι υπεύθυνος για την διαχείριση των sockets, την αποστολή και παραλαβή μηνυμάτων, την ενθυλάκωση των λειτουργιών για την διαδικτύωση, είναι ο πυρήνας της υποβιβλιοθήκης και επεκτείνεται από τον client και τον server οι οποίοι προσθέτουν λειτουργίες.

## 4.3 Υλοποίηση

### 4.3.1 Τρόπος χρήσης

Η διαδικασία χρήσης περιγράφεται στο διάγραμμα 4.2. Η χρήση πρέπει να είναι εξορθολογιστική και ξεκάθαρη για εύκολη και γρήγορη ανάπτυξη και περιοριστική για αποφυγή σφαλμάτων.

### 4.3.2 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.

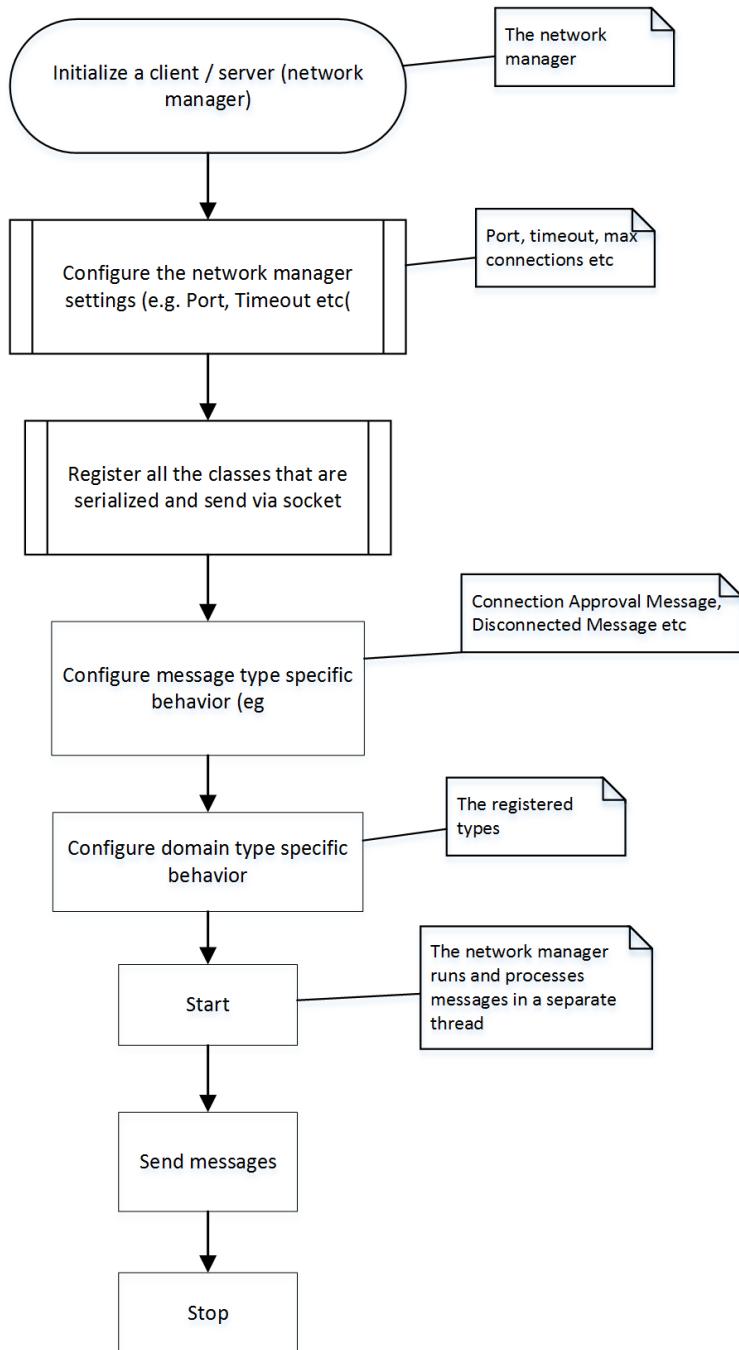
**Networking** Διαχειρίζεται τις συνδέσεις των χρηστών, τα sockets και ο, τι έχει να κάνει με διασύνδεση.

**Auditing** Αφαιρετικό επίπεδο για logging. Ο χρήστης μπορεί να ενσωματώσει τη δική του λογική παρακολούθησης μηνυμάτων.

**Infrastructure** Το επίπεδο αυτό περιέχει επαναχρησιμοποιήσιμο 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.

### 4.3.3 Ανταλλαγή πακέτων

Το serialization των μηνυμάτων πρέπει να είναι ντετερμινιστικό ώστε να αναγνωρίζεται ο τύπος του μηνύματος και η κλάση την οποία αντιπροσωπεύει και ελαφρύς ώστε να μην επιβαρύνεται το δίκτυο με επιπλέον δεδομένα. Ο ενσωματωμένος serializer χρησιμοποιεί reflection για να αναγνωρίσει τα primitive properties της κλάσης. Για προχωρημένο serialization περίπλοκων τύπων, ο χρήστης έχει τη δυνατότητα να συμπεριλάβει τον δικό του serializer με την υλοποίηση του IPackageSerializer interface και την καταχώρησή του σημειώνοντας την κλάση με το PackageSerializerAttribute και τον τύπο του serializer.



**Διάγραμμα 4.2:** Διάγραμμα χρήσης του υποσυστήματος δικτύωσης

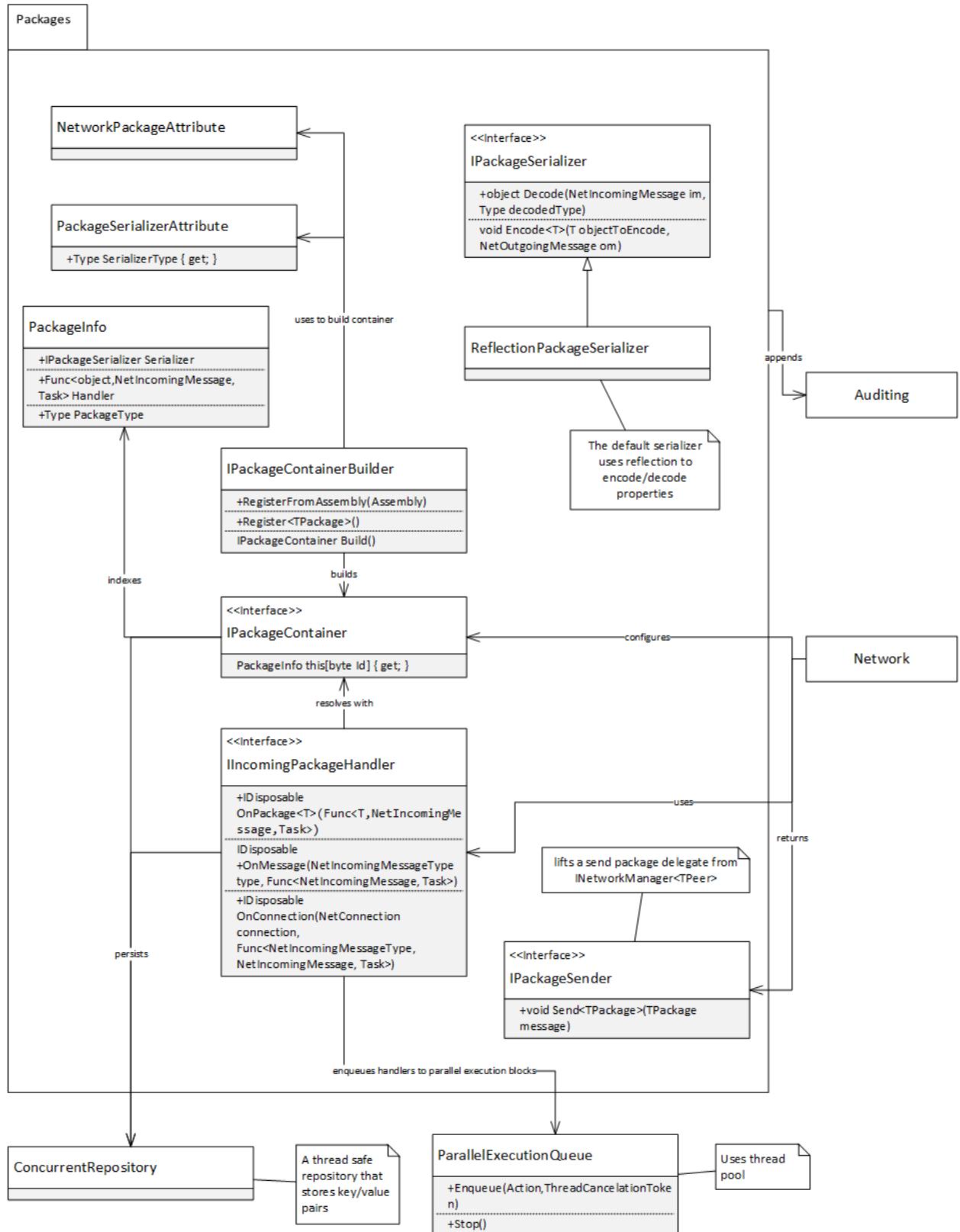
Κατά την προετοιμασία του network manager, ο χρήστης καλείται να καταχωρίσει στο container ποιες κλάσεις θα χρησιμοποιηθούν κατά την ανταλλαγή μηνυμάτων. Οι κλάσεις που προορίζονται για serialization σημειώνονται με κάποιο Attribute και η καταχώριση μπορεί να γίνει δυναμικά με reflection. Όταν τελειώσει η καταχώριση, το container χρησιμοποιώντας ένα ντετερμινιστικό 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 εκτελέσιμο κώδικα υπό μορφή function ο οποίος εκτελείται ασύγχρονα κατά την παραλαβή κάποιου μηνύματος-κλάσης, τύπου μηνύματος, ή συμβάντος συγκεκριμένης σύνδεσης.

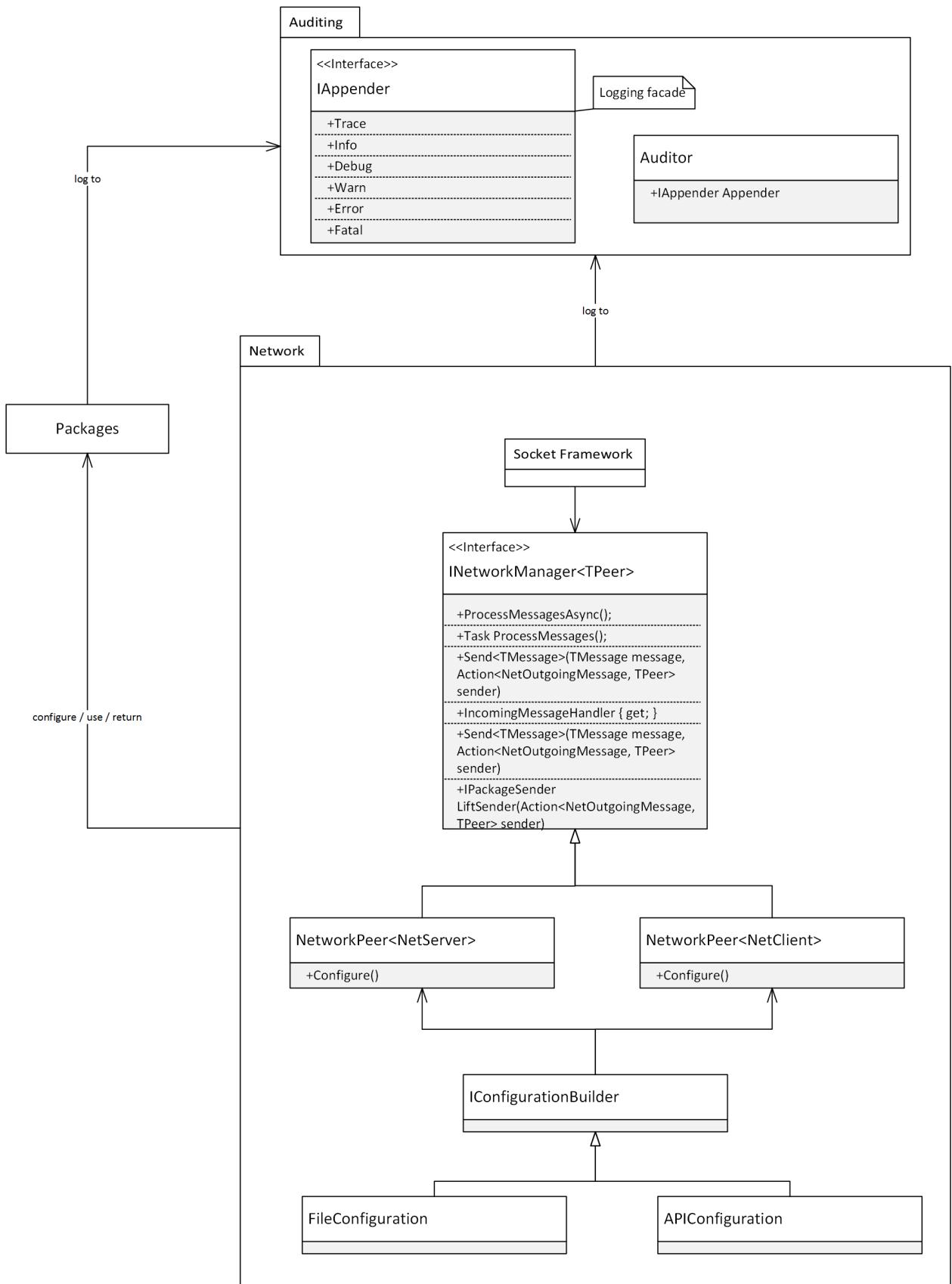
Στο UML διάγραμμα 4.3 παρουσιάζεται το υποσύστημα διαχείρισης πακέτων.

#### **4.3.4 Διαχείριση δικτύωσης**

Οι ρυθμίσεις του network manager κτίζονται από αφαιρέσεις. Η χρήση πρωτοκόλλων και τεχνικών γίνεται με άγνοια της υλοποίησης και των λεπτομερειών. Η σχεδίαση επιτρέπει την αποστολή μηνύματος ανεξαρτήτου πρωτοκόλλου, τρόπου 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 χρήσης, και να τους κρύψει πίσω από functions. Η αρχιτεκτονική του networking module παρουσιάζεται στο UML διάγραμμα 4.4



Διάγραμμα 4.3: Υποσύστημα διαχείρισης πακέτων



**Διάγραμμα 4.4:** Μονάδα διαχείρισης της δικτύωσης

### 4.3.5 Παραδείγματα API

Στα παρακάτω παραδείγματα, γίνεται μια σύντομη επισκόπηση του API.

- Κλάσεις που προορίζονται για ανταλλαγή μηνυμάτων

**Παράδειγμα κώδικα 4.1:** Attribute πακέτων

```

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

- Ρύθμιση και εξατομίκευση 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:** Χειρισμός εισερχόμενων τύπων-πακέτων

```
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 secret and encrypted password")
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:** Χειρισμός εισερχόμενων μηνυμάτων

```

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

```

- Αποστολή μηνύματος-κλάσης

**Παράδειγμα κώδικα 4.7:** Αποστολή μηνύματος-κλάσης

```

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:** Lift αποστολέα μηνυμάτων

```

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

```

```

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

```

- Διαγραφή χειριστή

**Παράδειγμα κώδικα 4.9:** Διαγραφή χειριστή

```

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*

## 4.4 Testing

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

**Παράδειγμα κώδικα 4.10:** Δοκιμαστικότητα εισερχόμενου πακέτου

```

1
2 [TestMethod]
3 public void Incoming_Package_GetsHandled()
4 {
5     // Arrange
6     var client = new NetworkCliient(
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 το πώς η εφαρμογή ανταποκρίνεται σε περιβάλλον κακής διαδικτύωσης και να αναπτυχθούν τεχνικές πρόβλεψης δικτύου (network prediction) για εξομάλυνση της κίνησης.

## Κεφάλαιο 5

### Διαγνωστικά και Αποσφαλμάτωση

Η ανάπτυξη του λογισμικού πρέπει να παρακολουθείται συνεχώς. 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. Ακόμη και στις τελικές εκδόσεις όμως υπάρχει η πιθανότητα σφάλματος.

#### 5.1 Logging and Tracing

Ένα logfile είναι ένα αρχείο το οποίο περιέχει γεγονότα τα οποία συμβαίνουν κατά την εκτέλεση ενός λογισμικού. Logging ονομάζουμε τη διαδικασία καταγραφής μηνυμάτων σε ένα 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 το οποίο οδηγεί στον τερματισμό του παιχνιδιού, πρέπει να δημιουργηθεί ένα crash report και να σταλεί στους προγραμματιστές για να εντοπίσουν το σφάλμα. Το crash report πρέπει να περιλαμβάνει:

- Το στιγμότυπο του χρόνου σε UTC.
- Το επίπεδο στο οποίο παρουσιάστηκε το σφάλμα.
- Την τοποθεσία του παίχτη.

- Την κατάσταση του παιχτη.
- Τα τρέχον gameplay scripts.
- Exception και stack trace.
- Την κατάσταση κατανομής μνήμης.
- Στιγμιότυπο οθόνης.

## 5.2 Time Benchmarking

Ο time ruler προσφέρει δυναμική ανάλυση του χρόνου εκτέλεσης και των ενημερώσεων με απόδοση πραγματικού χρόνου στην οθόνη με παραμετροποιήσιμα ονόματα και χρώμα μεταξύ της έκτασης υπολογισμού του. Ο time ruler παίρνει στιγμιότυπα του χρόνου κατά το χρόνο εκτέλεσης και τα αποδίδει ως 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 εναλλαγή. Στο παράδειγμα 5.1 παρουσιάζεται ένα απλό παράδειγμα χρήσης του time ruler για ανάλυση αλγορίθμου.

**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].Benchmark)
6 {
7     //benchmarking logic here
8 }
```

## 5.3 Debug Drawing API

Μία οντότητα η οποία αποδίδεται στην οθόνη περιέχει περισσότερες πληροφορίες από αυτό που δείχνει. Μια οντότητα αν συμμετέχει στη προσομοίωση φυσικής έχει τα-

χύτητα 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 διαγνωστικών 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 επιτυγχάνεται με τη χρήση των conditionals τα οποία χρησιμοποιούν οδηγίες προεπεξεργαστή.

Το υποσύστημα προσφέρει δυνατότητες απόδοσης για τα παρακάτω:

- 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. Μερικά από αυτά είναι τα παρακάτω:

- 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 και profiling blocks με αναγνώσιμα ονόματα.
- Ιεραρχικό Profiling.
- Εξαγωγή δεδομένων 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 υποσυστημάτων των οποίων αναφέρθηκαν στηρίζεται σε γνώση εις βάθους διάφορων τεχνολογιών όπως OpenGL, SDL κλπ καθώς και σε γνώση ιδιαίτεροτήτων πλατφορμών, καρτών γραφικών κλπ. Κάποια από τα υποσυστήματα τα οποία δεν αναφέρθηκαν είναι τα παρακάτω:

- Το υποσύστημα διαχείρισης ήχου.
- Το gameplay system το οποίο περιλαμβάνει τις λειτουργίες των μηχανισμών του παιχνιδιού.
- Το camera system το οποίο χειρίζεται τις κάμερες του παιχνιδιού.
- Το rendering system το οποίο βασίζεται σε καλή κατανόηση μαθηματικών, των σταδίων της κάρτας γραφικών (rendering pipeline) και σε πολύ χαμηλού επιπέδου βελτιστοποίησεις για τη βέλτιστη χρήση του κύκλου του επεξεργαστή, της μνήμης και της κάρτας γραφικών.

### 6.1 Επεκτασιμότητα

Η μηχανή προσφέρει δυνατότητες επεκτασιμότητας μέσω plugins. Ος plugin, ορίζεται ένα σύστημα συστατικών κάποιου λογισμικού που προσθέτει ιδιαίτερες δυνατότητες σε ένα μεγαλύτερο λογισμικό. Ένα εξειδικευμένο είδος plugin είναι το addon και περιλαμβάνει επεκτάσεις ή οπτικά θέματα. Η επέκταση μέσω plugins προσφέρει τα παρακάτω πλεονεκτήματα

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

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

## 6.2 Συμπεράσματα

Με ένα εργαλείο όπως μια μηχανή γραφικών, ο σχεδιαστής παιχνιδιών απαλλάσσεται από το βάρος της σχεδίασης, τις τεχνικές λεπτομέρειες των πλατφορμών και εστιάζει καθαρά στην ιδέα του. Προσφέρονται πολλές λειτουργίες οι οποίες χρησιμοποιούνται για να κτιστούν παιχνίδια εύκολα, γρηγορότερα και με λιγότερη ταλαιπωρία. Λόγω της γενικότητας όμως του εργαλείου, είναι δύσκολο να αντιμετωπίσει όλα τα προβλήματα της ανάπτυξης παιχνιδιών. Μέσω του ανοιχτού κώδικα και τις διάφορες δυνατότητες επέκτασης, όπως τα plugins, το εργαλείο μπορεί να επεκταθεί είτε από εισηγήσεις των χρηστών, είτε κατευθείαν από αυτούς. Αυτό είναι απαραίτητο γιατί ο σχεδιασμός και η ανάπτυξη των υποσυστημάτων των μηχανών γραφικών απαιτεί συνεργασία πολλών εξειδικευμένων ειδικοτήτων και το οποίο επιτυγχάνεται είτε μέσω εταιριών, είτε μέσω της κοινότητας του ανοικτού κώδικα.

## Βιβλιογραφία

- [1] ARM. OpenGL ES application development guide for the Mali GPU, 2007-2009.
- [2] Alex Champandard. Understanding behavior trees. *AiGameDev.com*, 6, 2007.
- [3] Christer Ericson. *Real-Time Collision Detection*. CRC Press, Inc., Boca Raton, FL, USA, 2004.
- [4] Thomas Erl. *SOA Design Patterns*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2009.
- [5] T. Fullerton. *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*. Taylor & Francis, 2008.
- [6] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
- [7] Arthur Gill et al. Introduction to the theory of finite-state machines. 1962.
- [8] J. Gregory. *Game Engine Architecture*. AK Peters Series. Taylor & Francis, 2009.
- [9] Jonathan S. Harbour. *Game Programming All in One*. Thomson Course Technology, 2007.
- [10] Raph Koster. *A Theory of Fun for Game Design*. Paraglyph Press, 2004.
- [11] Guy W. Lecky-Thompson. *Fundamentals of Network Game Development*. Course Technology, 2009.
- [12] Sanjay Madhav. *Game Programming Algorithms and Techniques*. Addison-Wesley, 2014.

- [13] 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.
- [14] B. Suits and T. Hurka. *The Grasshopper: Games, Life and Utopia*. Broadview encore editions. Broadview Press, 2005.
- [15] 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, 9, 11, 25, 42, 50, 65

**B-Tree** Το B-tree είναι μια αυτοεξισορροπούμενη δενδροειδή δομή δεδομένων που διατηρεί τα δεδομένα ταξινομημένα και επιτρέπει αναζητήσεις, διαδοχική πρόσβαση, εισαγωγές και διαγραφές σε λογαριθμικό χρόνο. . 41, 45

**Common Intermediate Language** Η Common Intermediate Language βρίσκεται στα χαμηλότερα επίπεδα αναγνώσιμων από ανθρωπο γλωσσών προγραμματισμού, σχεδιάστηκε από τις προδιαγραφές της Common Language Infrastructure (CLI) και χρησιμοποιείται από το .NET και το Mono.. 10

**dpi** Το DPI (Dots per inch) είναι μέτρο της πυκνότητας των τελειών που μπορούν να τοποθετηθούν σε μια γραμμή στο μήκος μιας ίντζας (2.54 cm).. 40

**fps** Το FPS (frames per second), είναι η συχνότητα (ρυθμός) στην οποία μία συσκευή απεικόνισης εμφανίζει διαδοχικές εικονές, τα οποία ονομάζονται πλαίσια.. 26, 27, 72

**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. H/Y) 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).. 28

**IP** Μία διεύθυνση IP (Ip address - Internet Protocol address), είναι ένας μοναδικός αριθμός που χρησιμοποιείται από συσκευές για τη μεταξύ τους αναγνώριση και συνεννόηση σε ένα δίκτυο υπολογιστών που χρησιμοποιεί το Internet Protocol standard. . 56

**MDA** Το MDA είναι μια επίσημη προσέγγιση για την κατανόηση του τι είναι ένα παιχνίδι, και επιχειρεί να γεφυρώσει το χάσμα μεταξύ του σχεδιασμού, ανάπτυξης κριτικής και τεχνικής έρευνας του παιχνιδιού. . 21

**mocks** Στον αντικειμενοστραφή προγραμματισμό, τα mock αντικείμενα προσομοιώνουν αντικείμενα που μιμούνται τη συμπεριφορά των πραγματικών αντικειμένων με ελεγχόμενες παραμέτρους . Ένας προγραμματιστής συνήθως δημιουργεί ένα mock αντικείμενο για να ελέγξει τη συμπεριφορά κάποιου άλλου αντικειμένου.. 68

**modding** Το modding αναφέρεται στην τροποποίησης του υλικού ή του λογισμικού, για να εκτελέσει μια λειτουργία η οποία δεν είχε αρχικά σχεδιαστεί ή που προορίζονται από τον σχεδιαστή.. 15

**OpenGL** Η OpenGL (Open Graphics Library) είναι μια διασύνδεση προγραμματισμού εφαρμογών ανεξαρτήτου γλώσσας ή πλατφόρμας για την απόδοση δυανισματικών 2D και 3D γραφικών. Χρησιμοποιεί τεχνικές επιτάχυνσης υλικού μέσω της μονλάδας επεξεργασίας γραφικών (GPU).. 15, 74

**OpenGL ES** Η OpenGL ES (Open GL for Embedded Systems) είναι υποσύνολο της OpenGL και χρησιμοποιείται σε συστήματα όπως κινητά, tablets, κονσόλες κλπ.. 15

**port** Στη δικτύωσης υπολογιστών, το port (θύρα) είναι μια παράμετρος της επικοινωνίας σε ένα λειτουργικό σύστημα.. 56

**SDK** Ένα SDK (Software Development Kit) είναι ένα σύνολο εργαλείων ανάπτυξης που επιτρέπουν σε έναν προγραμματιστή να δημιουργήσει λογισμικό εφαρμογών για ένα συγκεκριμένο πακέτο λογισμικού, πλατφόρμα, παιχνιδομηχανή, λειτουργικό σύστημακ κλπ.. 15

**SOLID** To SOLID (single responsibility, open-closed, Liskov substitution, interface segregation and dependency inversion) είναι ένα ακρώνυμο το οποίο θέσπισε ο Michael Feathers για τις πέντε πρώτες αρχές (first five principles) του Robert C Martin στις αρχές του 2000. Όταν ένας προγραμματιστής χρησιμοποιήσει αυτές τις αρχές, θα πιο πιθανό είναι ότι το σύστημά το οποίο θα δημιουργήσει θα είναι εύκολο να διατηρηθεί, να επεκταθεί και με λιγότερα σφάλματα.. 11

**UML** H UML (Unified Modeling Language) πλέον είναι η πρότυπη γλώσσα μοντελοποίησης στη μηχανική λογισμικού. Χρησιμοποιείται για τη γραφική απεικόνιση, προσδιορισμό, κατασκευή και τεκμηρίωση των στοιχείων ενός συστήματος λογισμικού. Μπορεί να χρησιμοποιηθεί σε διάφορες φάσεις ανάπτυξης, από την ανάλυση απαιτήσεων ως τον έλεγχο ενός ολοκληρωμένου συστήματος.. 13, 31, 34, 42, 48, 51, 62

**XML** H XML (Extensible Markup Language) είναι μία γλώσσα σήμανσης, που περιέχει ένα σύνολο κανόνων για την ηλεκτρονική κωδικοποίηση κειμένων. Ορίζεται, κυρίως, στην προδιαγραφή XML 1.0 (XML 1.0 Specification), που δημιουργήσε ο διεθνής οργανισμός προτύπων W3C (World Wide Web Consortium), αλλά και σε διάφορες άλλες σχετικές προδιαγραφές ανοιχτών προτύπων.. 21, 45

# Κατάλογος διαγραμμάτων

2.1	Τυπική αρχιτεκτονική μηχανής γραφικών . . . . .	17
3.1	Κύκλος ζωής του πυρήνα . . . . .	26
3.2	Βρόγχος παιχνιδιού . . . . .	27
3.3	Διάγραμμα ακολουθίας του υποσυστήματος διαχείρισης οθονών . . . . .	29
3.4	UML του υποσυστήματος διαχείρισης οθονών . . . . .	31
3.5	Αρχιτεκτονική υποσυστήματος διαχείρισης εισόδων . . . . .	35
3.6	Υποσύστημα φυσικής και εντοπισμού συγκρούσεων . . . . .	40
3.7	B-Tree διεπαφής χρήστη . . . . .	41
3.8	Δομή διεπαφής χρήστη . . . . .	43
3.9	UML του συστήματος ανθρώπινης διεπαφής . . . . .	44
3.10	Κύκλος ζωής διεπαφής χρήστη . . . . .	45
3.11	UML του διαχειριστή πόρων πραγματικού χρόνου . . . . .	48
3.12	Δέντρο συμπεριφορών . . . . .	50
3.13	UML του υποσυστήματος των state machines . . . . .	52
3.14	Παράδειγμα state machine . . . . .	52
4.1	Διάγραμμα ακολουθίας του υποσυστήματος δικτύωσης . . . . .	56
4.2	Διάγραμμα χρήσης του υποσυστήματος δικτύωσης . . . . .	61
4.3	Υποσύστημα διαχείρισης πακέτων . . . . .	63
4.4	Μονάδα διαχείρισης της δικτύωσης . . . . .	64

# Κατάλογος παραδειγμάτων κώδικα

3.1	Εξατομικευτής εναλλαγής οθονών . . . . .	30
3.2	Παράδειγμα API του υποσυστήματος διαχείρισης οθονών-σκηνών . .	32
3.3	Παράδειγμα καταχώρησης ακροατών στο υποσύστημα εισόδων . . .	35
3.4	Behavior tree builder . . . . .	51
3.5	State machine API . . . . .	54
4.1	Attribute πακέτων . . . . .	65
4.2	Ρύθμιση server . . . . .	65
4.3	Καταγραφή κινήσεων . . . . .	66
4.4	Απάντηση κατά την παραλαβή μηνύματος . . . . .	66
4.5	Χειρισμός εισερχόμενων τύπων-πακέτων . . . . .	66
4.6	Χειρισμός εισερχόμενων μηνυμάτων . . . . .	67
4.7	Αποστολή μηνύματος-κλάσης . . . . .	67
4.8	Lift αποστολέα μηνυμάτων . . . . .	67
4.9	Διαγραφή χειριστή . . . . .	68
4.10	Δοκιμαστικότητα εισερχόμενου πακέτου . . . . .	68
5.1	Time Ruler . . . . .	71

Η εργασία αυτή στοιχειοθετήθηκε με το πρόγραμμα **X<sub>E</sub>ΛΑΤΕΧ**. Για τη στοιχειοθέτηση της βιβλιογραφίας χρησιμοποιήθηκε το πρόγραμμα **bibtex**. Οι γραμματοσειρές που χρησιμοποιήθηκαν είναι οι **Times New Roman** και **Courier New**.