ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΤΕ

ΑΝΑΠΤΥΞΗ ΕΡΓΑΛΕΙΟΥ ΓΙΑ ΤΗΝ ΣΧΕΔΙΑΣΗ ΠΑΙΧΝΙΔΙΩΝ ΜΕ ΤΗΝ ΒΟΗΘΕΙΑ ΥΠΟ-ΛΟΓΙΣΤΗ

Πτυχιακή εργασία του

Γιώργου Μιχαηλίδη

Επιβλέπων: Δρ. Νικόλαος Πεταλίδης. Επιστημονικός Συνεργάτης

ΣΕΡΡΕΣ, ΦΕΒΡΟΥΑΡΙΟΣ 2016

Υπεύθυνη δήλωση

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

Σύνοψη

Τα πρώτα ηλεκτρονικά παιχνίδια είχαν γραφτεί εξ'ολοκλήρου σε υλισμικό. Από τότε, οι κάρτες γραφικών και οι μικροεπεξεργαστές βελτιώθηκαν, δημιουργήθηκαν κονσόλες φτιαγμένες αποκλειστικά για ηλεκτρονικά παιχνίδια, με ειδικά χειριστήρια τα οποία σου προσφέρουν διαφορετικές εμπειρίες. Η διαδικασία ανάπτυξης λογισμικού είναι ακριβή και ο σχεδιασμός γίνεται όλο πιο σύνθετος και περίπλοκος. Τα έργα γίνονται όλο πιο απαιτητικά και δαπανηρά. Δημιουργήθηκε η ανάγκη για ένα εργαλείο το οποίο να παρέχει ένα ομοιογενές περιβάλλον για την ανάπτυξη σύνθετων έργων. Ένα CASE (Computer Aided Software Engineering) tool είναι ένα λογισμικό-εργαλείο το οποίο απλοποιεί τον κύκλο ανάπτυξης ενός λογισμικού. Στο τομέα του σχεδιασμού παιχνιδιών το πιο διαδεδομένο CASE tool είναι η μηχανή γραφικών. Μια μηχανή γραφικών είναι μια σουίτα 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.

Περιεχόμενα

Κατάλογος πινάκων

Κατάλογος διαγραμμάτων

Κατάλογος ορισμών

Πρόλογος

Σκοπός Σκοπός της πτυχιακής είναι να εξηγήσει θεωρητικά και πρακτικά τα κομμάτια που απαρτίζουν μια τυπική μηχανή γραφικών και πώς συνδέονται αρχιτεκτονικά μεταξύ τους, μαζί με παραδείγματα και οδηγίες για την επέκτασή τους. Για το κομμάτι του rendering, δεν έγινε απευθείας με προτόκολο που επικοινωνεί με την κάρτα γραφικών, αλλά με την ανοικτού λογισμικού βιβλιοθήκη monogame η οποία είναι και cross-platform, δηλαδή ανάλογα με το περιβαλλον ανάπτυξης χρησιμοποιεί το ανάλογο προτόκολο για επικοινωνία με την κάρτα γραφικών, για παράδειγμα για Linux OS χρησιμοποιεί OpenGL και για Android OpenGL ES, για να επικεντρωθεί στο rendering γενικά και όχι για συγκεκριμένη πλατφόρμα. Τα παραδείγματα αξιοποιούν object-oriented και functional paradigms και είναι γραμμένα σε C#.

Θεμελίωση Οι μηχανές γραφικών διαφέρουν με βάση τις λεπτομέριες αρχιτεκτονικής και υλοποίησης, αλλά τα πρότυπα σχεδίασης είναι καθολικά. Πρακτικά, οι μηχανές γραφικών απαρτίζονται από τις ίδιες βασικές έννοιες. Πολλά βιβλία έχουν γραφτεί τα οποία εστιάζουν στην κάθε έννοια ξεχωριστά αλλά ελάχιστα για το πως αυτά τα στοιχεία επικοινωνούν και αλληλεπιδρούν μεταξύ τους. Η εργασία εστιάζει στην αρχιτεκτονική των μηχανών, για το πώς οι ομάδες είναι οργανωμένες για να δουλεύουν μεταξύ τους, ποια συστήματα και μοτίβα επαναλαμβάνονται στη δημιουργία μηχανών, ποιες είναι οι απαιτήσεις για το κάθε μεγάλο υποσύστημα της μηχανής, ποια συστήματα είναι αγνωστικιστικά σε παιχνίδια ή σε είδη παιχνιδιών και ποια είναι συγκεκριμένα, και πότε σταματά η μηχανή και ξεκινά η υλοποίηση του παιχνιδιού.

Επιπρόσθετα, θα γίνει σύγκριση με άλλες δημοφιλείς μηχανές γραφικών και βιβλιοθηκών τεχνικές για configuration management, versioning και διάφορα συστήματα ανάπτυξης.

Ευχαριστίες

Ευχαριστίες (στο μπαμπά, στη μαμά, κτλ)

Κεφάλαιο 1

Case Tools

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

1.1 Κοινές λειτουργίες

- 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).

1.2 Γιατί;

- Γρήγορη εγκατάσταση
- Εξοικονόμηση χρόνου μειώνοντας τον χρόνο στον προγραμματισμό και στις δοκιμές.
- Οπτικοποίηση του κώδικα και της ροής δεδομένων
- Βέλτιστη χρήση των διαθέσιμων πόρων.
- Ανάλυση, ανάπτυξη και σχεδιασμό με ενιαίες μεθοδολογίες.
- Δημιουργία και τροποποίηση τεκμηρίωσης (documentation)
- Αποτελεσματική μεταφορά πληροφοριών ανάμεσα στα διάφορα εργαλεία
- Γρήγορη δημιουργία λογισμικού.

1.3 Χρήση

- Για να διευκολυνθεί η μεθοδολογία σχεδιασμού.
- Για Rapid Application Development
- Testing
- Documentation
- Project Management
- Μειωμένο κόστος συντήρησης
- Αύξηση της παραγωγικότητας:

Η Αυτοματοποίηση των διαφόρων δραστηριοτήτων των διαδικασιών ανάπτυξης και διαχείρισης του συστήματος αυξάνει την παραγωγικότητα της ομάδας ανάπτυξης.

1.4 Χαρακτηριστικά ενός καλού case tool

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

Κεφάλαιο 2

Game Engineering

2.1 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 πραγματκό 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 τον παιχνιδιών μέσω μιας μηχανής γραφικών γίνεται εύκολα, γρήγορα και οδηγούμενη απο δεδομένα (data driven) ούτως ώστε οι δημιουργοί παιχνιδιών να μπορούν να ασχολούνται με τις λεπτομέρειες του παιχνιδιού τους. Οι μηχανές αναπτύσσονται από ομάδες που απαρτίζονται όχι μόνο από προγραμματιστές, αλλά και απο μαθηματικούς, φυσικούς κλπ. Η κάθε υπο-ομάδα εστιάζει σε ένα συγκεκριμένο κομμάτι, όπως οι φυσικοί με τον εντοπισμό συγκρούσεων, και αρχιτέκτονες λογισμικού σχεδιάζουν το πως τα κομμάτια συνδέονται και αλληλεπιδρούν μεταξύ τους, χωρίς να τους απασχολούν οι λεπτομέριες σχεδίασης του κάθε κομματιού.

2.1.1 Ιστορία

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

2.2 Δομή μιας τυπικής ομάδας ανάπτυξης παιχνιδιών

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

2.2.1 Μηχανικοί

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

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

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

2.2.2 Artists

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

- Concept artists οι οποίοι σχεδιάζουν σκίτσα και πίνακες τα οποία παρέχουν στην ομάδα την εικόνα του τελικού παιχνιδιού. Παρέχουν οπτική καθοδήγηση στην ομάδα καθ' όλη τη διάρκει του κύκλου ανάπτυξης.

- 3D Modelers οι οποίοι είναι υπεύθυνοι για την τρισδιάστατη γεωμετρία του εικονικού κόσμου του παιχνιδιού. Απαρτίζονται από τους foreground modelers οι οποίοι σχεδιάζουν χαρακτήρες, οχήματα, οπλα και αντικείμενα του τρισδιάστατου κόσμου background modelers οι οποίοι σχεδιάζουν την στατικό περιβάλλον πχ κτήρια

- Texture artists οι οποίοι σχεδιάζουν τις δισδιάστατες εικόνες που καλύπτουν τα τρισδιάστατα μοντέλα

- Lighting artists που ορίζουν τις στατικές και δυναμικές πηγές φωτός και δουλεύουν με το χρώμα, την κατεύθυνση και την κατεύθυνση του φωτός.

- Animators οι οποίοι σχεδιάζουν την κίνηση των χαρακτήρων και των αντικειμένων

- Motion capture actors οι οποίοι παρέχουν ακατέργαστα δεδομένα κίνησης για να επεξεργαστούν οι animators και να τα ενσωματώσουν στο παιχνίδι.

- Sound designers οι οποίοι παράγουν τα εφέ και τη μουσική.

- Voice actors τους οποίους η φωνή ηχογραφείται και χρησιμοποιείται για τους χαρακτήρες στο παιχνίδι

2.2.3 Game Designers

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

2.2.4 Producers

Ο ρόλος του 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 ΑΑΑ εταιριών, όπου υπάρχουν πολλά τμήματα και πολλοί εργαζόμενοι.

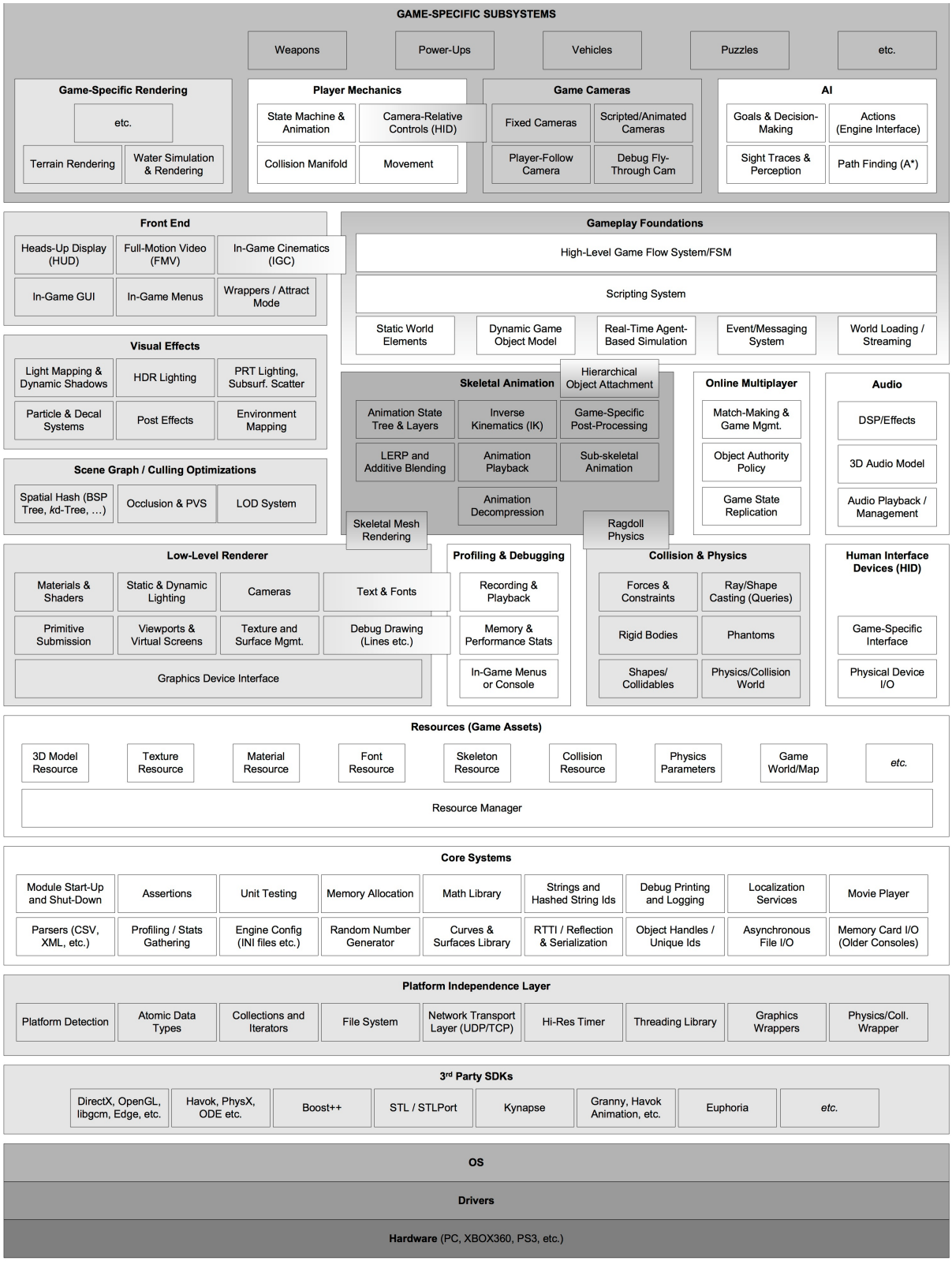
2.3 Τι είναι μια μηχανή γραφικών

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

2.4 Γιατί μηχανές γραφικών;

Η αφαίρεση πάντα βοηθούσε τον εγκέφαλο να λειτουργήσει καλύτερα και να κατανοήσει αλληλεπιδράσεις μεταξύ συστημάτων και περίπλοκες έννοιες. Οι μηχανές γραφικών απαλλάζουν τους γραφίστες και τους προγραμματιστές από τις τεχνικές λε-

πτομέριες, και εστιάζουν στην αισθητική και στο gameplay. Επίσης με την αποσύνδεση των συστημάτων έχουμε πιο προβλέψημη συμπεριφορά, επεκταστημότητα των υποσυστημάτων ως υποσυστήματα και ευκολη δοκιμαστικότητα.



Διάγραμμα 2.1: Game Engine Architecture

Κεφάλαιο 3

Ο πηρύνας της Μηχανής

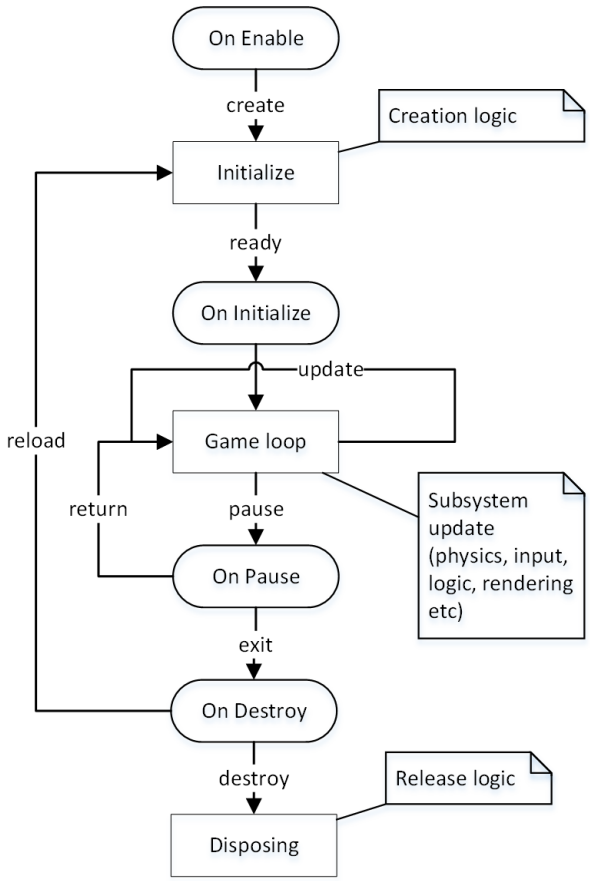
Ένα framework το οποίο επεκτείνεται και διακλαδώνεται σε πολλές υπο-βιβλιοθήκες, στηρίζεται στη θεμελιώση ενός framework core, του πηρύνα της βιβλιοθήκη. Το API του πηρύνα πρέπει να είναι εύκολο στην κατανόηση, να αποτελείται από αυτοεπεξηγούμενες αφαιρέσεις, οι οποίες εκθέτουν μόνο τα απολύτος απαραίτητα, ώστε οι υποβιβλιοθήκες που χρησιμοποιούν τον πηρύνα να μην δεσμεύονται σε υλοποιήσεις, να περιορίζει σε συγκεκριμένο pipeline χρήσης για αποφυγή απρόβλεπτων συμπεριφορών και να προσφέρει δυνατότητες επεκτασιμότητας. Tulach 2008

Theorem 1 (First test theorem). This is a test theorem.

3.1 Κύκλος ζωής πηρύνα

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

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



Διάγραμμα 3.1: Core Lifecycle

3.1.1 Game Loop

Το 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.

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

- το game loop στο οποίο ενημερώνονται όλα τα υποσυστήματα, physics, logic, dynamics κλπ
- rendering loop στο οποίο γίνονται οι κλήσεις στην κάρτα γραφικών και τρέχει στα 50-60 fps

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

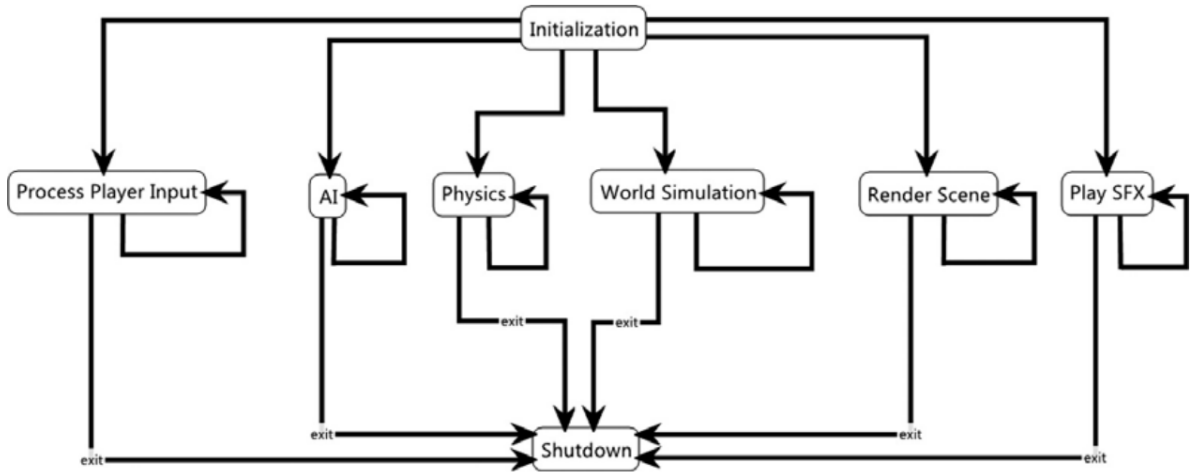
$$F = 1/T => F = 1000ms/60 => F = 16ms. \tag{3.1}$$

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

Στα συστήματα πραγματικού χρόνου, η έννοια της διάρκειας και του χρόνου πρέπει να χειρίζεται ως μια ξεχωριστή οντότητα. Ο χρόνος πρέπει να είναι ανεξάρτητος από τον πραγματικό χρόνο. Ένα animation το οποίο γίνεται render σε πραγματικό χρόνο, μπορεί να παίξει αντίστροφα ή με διπλάσια ταχύτητα, αν το χρονοδιάγραμμα στο οποίο ανταποκρίνεται, χειρίζεται τον χρόνο διαφορετικά. Όλα τα συστήματα ενημερώνονται γραμμικά συναρτήσει του χρόνου του οποίο παρέχεται ως είσοδος.

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



Διάγραμμα 3.2: game loops

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

3.1.3 Τεχνικές Ενημέρωσης Υποσυστημάτων

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

- Message Pumps: τα υποσυστήματα στέλνουν μηνύματα σε ένα message bus και τα συστήματα ενημερώνονται όταν εξυπηρετείται το μήνυμα. Παραμένουν άεργα εφόσον δεν υπάρχει μήνυμα προς εξυπηρέτηση.

- Call-back driven: τα υποσυστήματα παρέχουν call-back λειτουργικότητα. Δηλαδή τη δυνατότητα να θέσεις τι κώδικας θα εκτελεστεί κατά κάποιο συμβάν. Ένα συμβάν μπορεί να είναι η σύγκρουση μεταξύ δύο αντικειμένων. Ο χρήστης μπορεί να πει ότι όταν συμβεί σύγκρουση μεταξύ του παίχτη και του εχθρού, ο παίχτης θα χάσει ζωή.

3.2 ScreenSystem

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.

3.2.1 Απαιτήσεις

Ο χρήστης του συστήματος πρέπει να έχει πλήρη έλεγχο της κάθε οθόνης -σκηνής και να μπορεί να προσαρμόζει τη λογική και την φωτοαπόδοση ανάλογα. Εκτός από την ατομική λειτουργικότητα, η διαχείριση και εναλλαγή των οθονών-σκηνών πρέπει να γίνεται αυτόματα.

3.2.2 Καταστάσεις

Definition of screen states

3.3 Input System

3.3.1 Human Interface devices

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

- Polling: η κατάσταση κάποιον συσκευών (κυρίως της παλιάς σχολής) διαβάζεται ρωτώντας τη συσκευή για την κατάστασή της περιοδικά. Αυτό καταλήγει σε πλεονασμό γιατί ρωτάει πολλές φορές χωρίς να παίρνει απάντηση και με μικρή καθυστέρηση γιατί η αλλαγή κατάστασης μπορεί να γίνει μεταξύ ερωτήσεων.
- Interrupts (διακοπές): Οι συσκευές στέλνουν δεδομένα μόνο όταν αλλάξει η κατάσταση με κάποιο τρόπο. Ο χρήστης μπορεί γράψει κώδικα και να τον εγγράψει με τρόπο ώστε να εκτελείται μόνο όταν συμβεί κάποια αλλαγή κατάστασης.

3.3.2 Τύποι Input

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

3.3.3 Απαιτήσεις υποσυστήματος

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

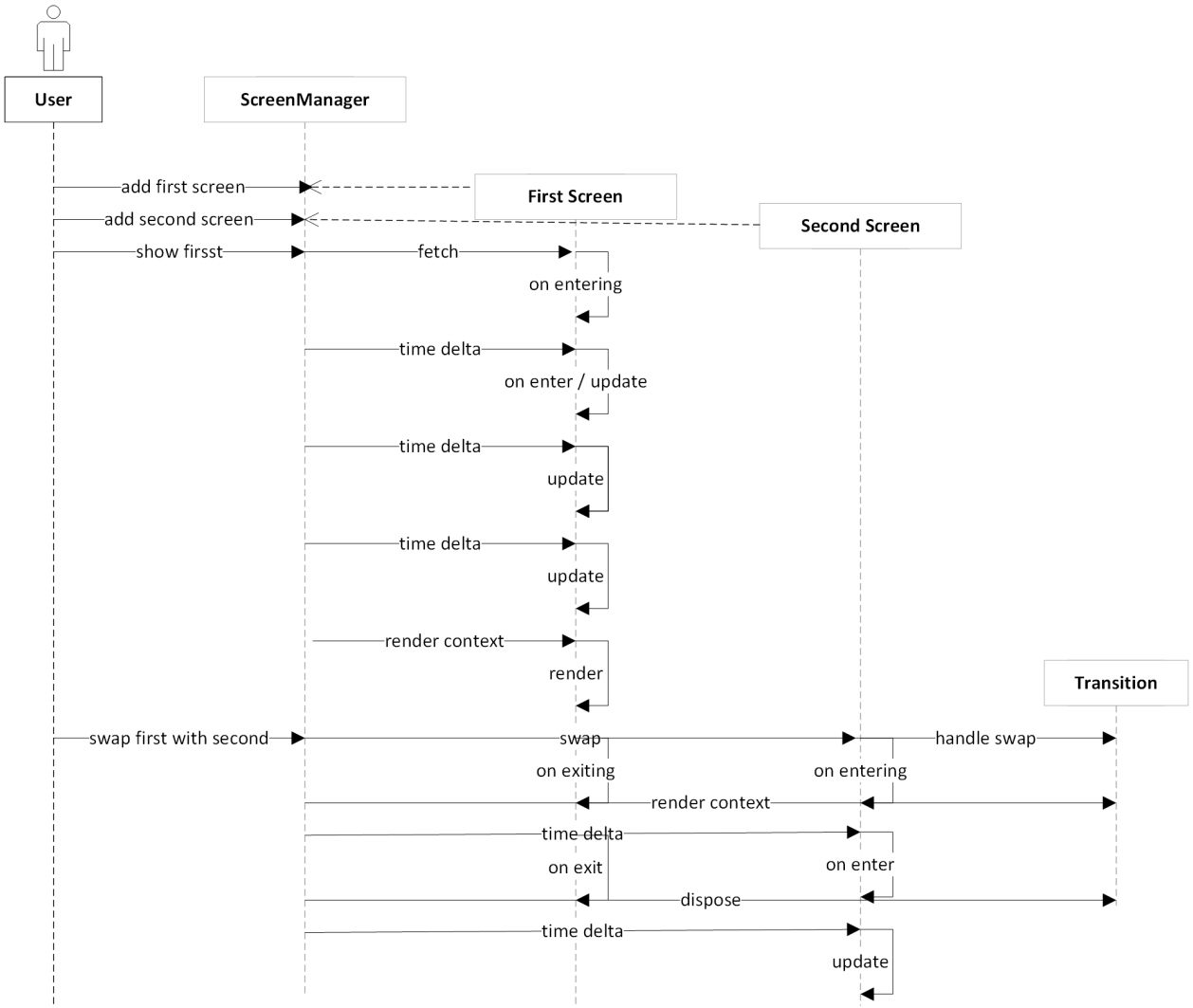
3.3.4 Observers-listeners

Το κάθε υποσύστημα διαχείρισης συσκευών ανθρώπινων διεπαφών αποτελείται από τον διαχειριστή για παράδειγμα τον MouseListener ή Keyboard listener, το οποίο χρησιμοποιεί τις βιβλιοθήκες του τρέχον λειτουργικού για την επιστολή συμβάντων των διεπαφών. Ανάλογα με την πλατφόρμα στην οποία έγινε compile δημιουργούνται οι αντίστοιχοι listeners μέσω του abstract factory. Ο χρήστης μπορεί να προσκολήσει κώδικα ο οποίος να εκτελείται ανά συμβαν π.χ. στη μετακίνηση του ποντικιού. Οι listeners ανάλογα με τις συνθήκες επιστολής, όπως το δέλτα του χρόνου επιστολής συμβάντος διπλού click ειναι 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, ανεξαρτή-

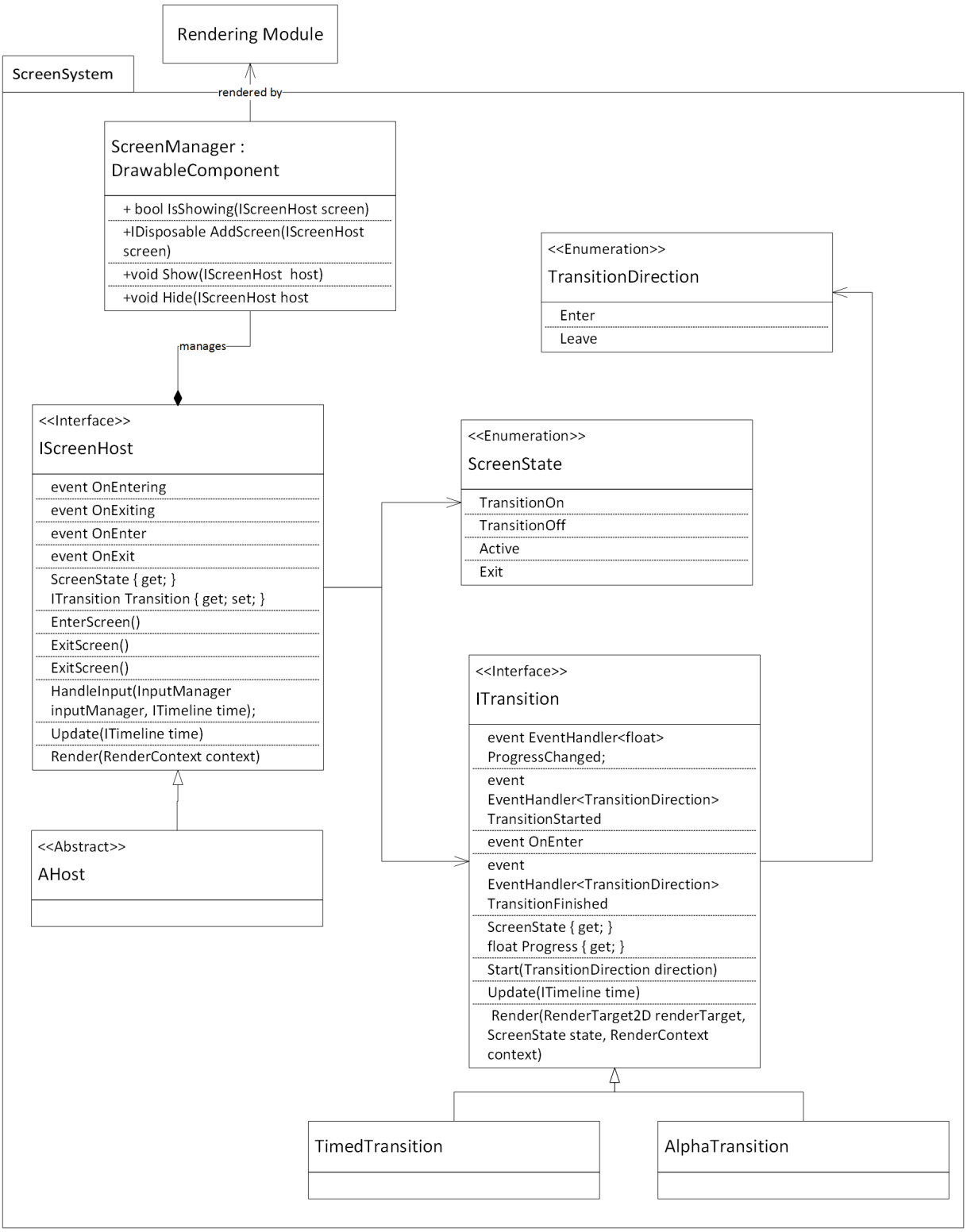
τως συμβάντος, και να προσκοληθεί στο initialization του lifecycle σε υποσυστήματα διεπαφών.

Listing 3.1: Παράδειγμα listener στο initialization

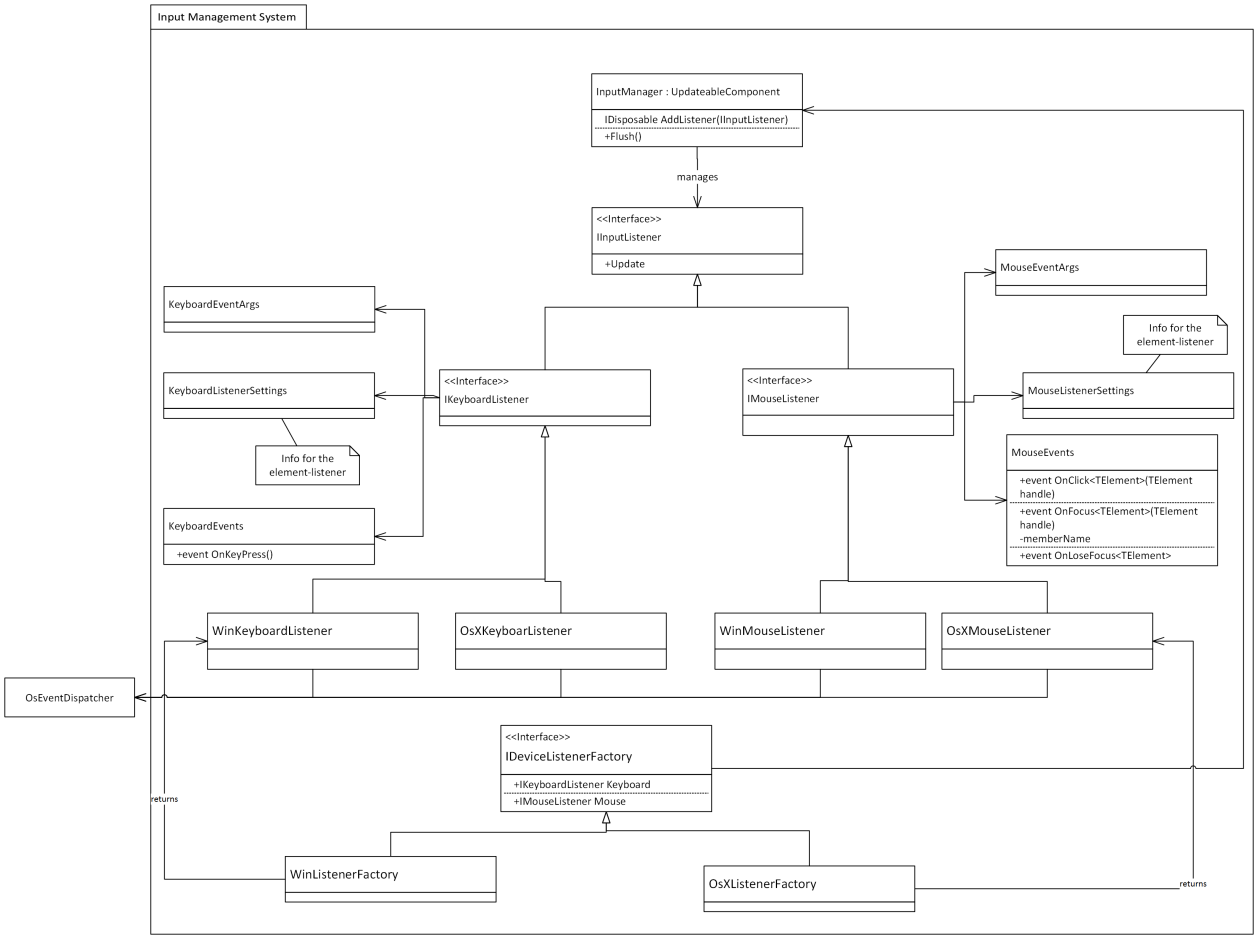
```
inputManager.AddListener(factory => factory.GamepadListener
{
        Settings = MouseSettings
                    .DoubleClickDelta(Timespan.FromSeconds
                        (0.2))
                        .DragDelta(Timespan.FromSeconds(0.2)),
        Events   = MouseEvents
                .OnLeftClick(sender,args) => this.Shoot())
                .OnLeftDoubleClick(sender,args) => this.Roll()
                    )
});

inputManager.AddListener(factory => factory.GamepadListener
{
        Settings = GamepadSettings.Default,
        Events   = GamepadEvents
                        .OnXButtonPress(sender,args) => this.
                            Shoot())
                        .OnYButtonPress(sender,args) => this.
                            Roll())
});
```



Διάγραμμα 3.3: screensystem sequence



Διάγραμμα 3.4: core screensystem



Διάγραμμα 3.5: core input architecture

Κεφάλαιο 4

Διαδικτύωση

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

4.1 Το πρόβλημα

4.1.1 Περιγραφή του προβλήματος

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

4.1.2 Κατανόηση του προβλήματος

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

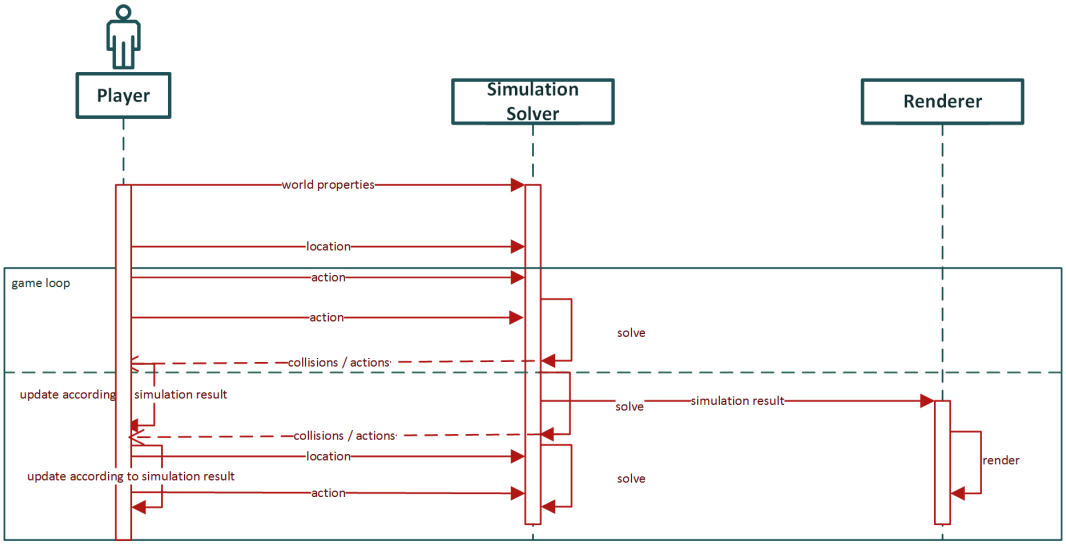
Κατά την ενημέρωση του κόσμου, ο προσομοιωτής χρησιμοποιώντας τους νόμους, τους κανόνες και τους μηχανισμούς που διέπουν τον κόσμο, παίρνει ως είσοδο τις οντότητες, τα ειδικά βάρη των ιδιοτήτων τους, την απόλυτη θέση τους στο σύστημα συντεταγμένων του κόσμου, και το χρονικό διάστημα της προσομοίωσης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

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

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

4.1.3 Εξαγωγή απαιτήσεων

Με βάση το πρόβλημα καταλήγουμε στο παρακάτω διάγραμμα ακολουθίας.



Διάγραμμα 4.1: Network Sequence

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

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

χρόνο.

4.2 Εκπόνηση σχεδίου

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

4.2.1 Επιλογή πρωτοκόλου

- TCP (transmission control protocol), είναι το πιο συχνά χρησιμοποιημένο πρωτόκολλο. Η διασύνδεση με TCP είναι αξιόπιστη και τα μηνύματα παραλαμβάνονται στη σειρά αποστολής. Η αξιοπιστία όμως έρχεται με ένα μικρό κόστος απόδοσης.

- UDP To UDP (user diagram protocol) δεν περιλαμβάνει την αξιοπιστία και την εγγύηση της αλληλουχίας μηνυμάτων. Η απουσία λειτουργιών όμως, το κάνει το πιο γρήγορο σε αποστολή μηνυμάτων πρωτόκολλο.

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

4.2.2 Επιλογή αρχιτεκτονικής δικτύου

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

- Client-server model: στο οποίο ο client απλά κάνει render και το μεγαλύτερο κομμάτι της λογικής και της προσωμοίωσης τρέχει στον server. Ο server στέλνει οδηγίες στον client για το τι να κάνει render και ο client απλά υπακούει.

- Client on top of server model: ο client είναι και server, δηλαδή οι μηχανές που έχουν τον client έχουν και τον server.

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

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

4.3 Σχεδίαση του framework

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

4.3.1 Έννοιες

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

4.3.2 Τύποι μηνυμάτων

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

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

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

4.3.3 NetworkManager

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

4.4 Υλοποίηση

4.4.1 Τρόπος χρήσης

Η διαδικασία χρήσης πρέπει να είναι εξορθολογιστική και ξεκάθαρη για εύκολη και γρήγορη ανάπτυξη και περιοριστική για αποφυγή bugs και προβλημάτων.

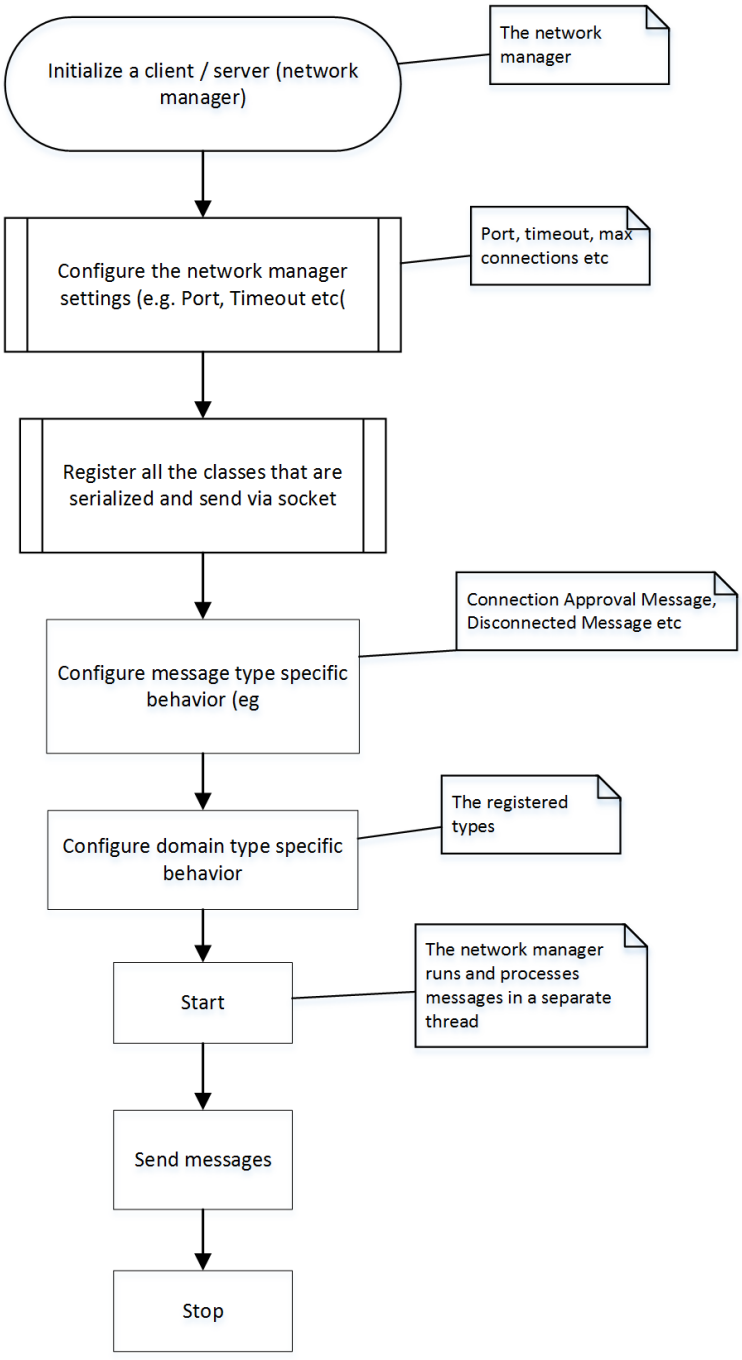
4.4.2 Αρχιτεκτονική

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

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

4.4.3 Packages Module

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



Διάγραμμα 4.2: Network Usage Diagram

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

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

4.4.4 Networking Module

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

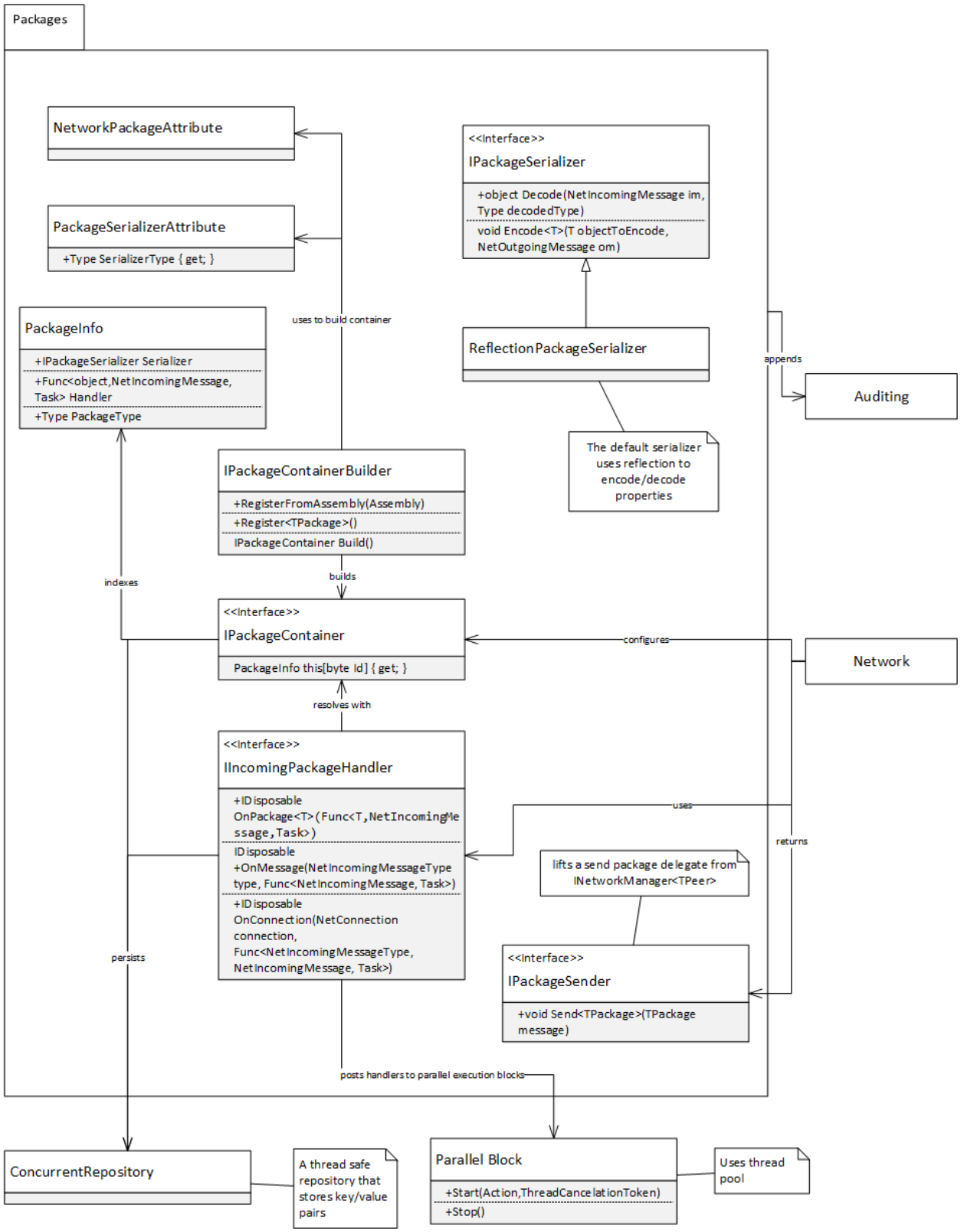
4.4.5 API

Συντομη επισκόπηση του API.

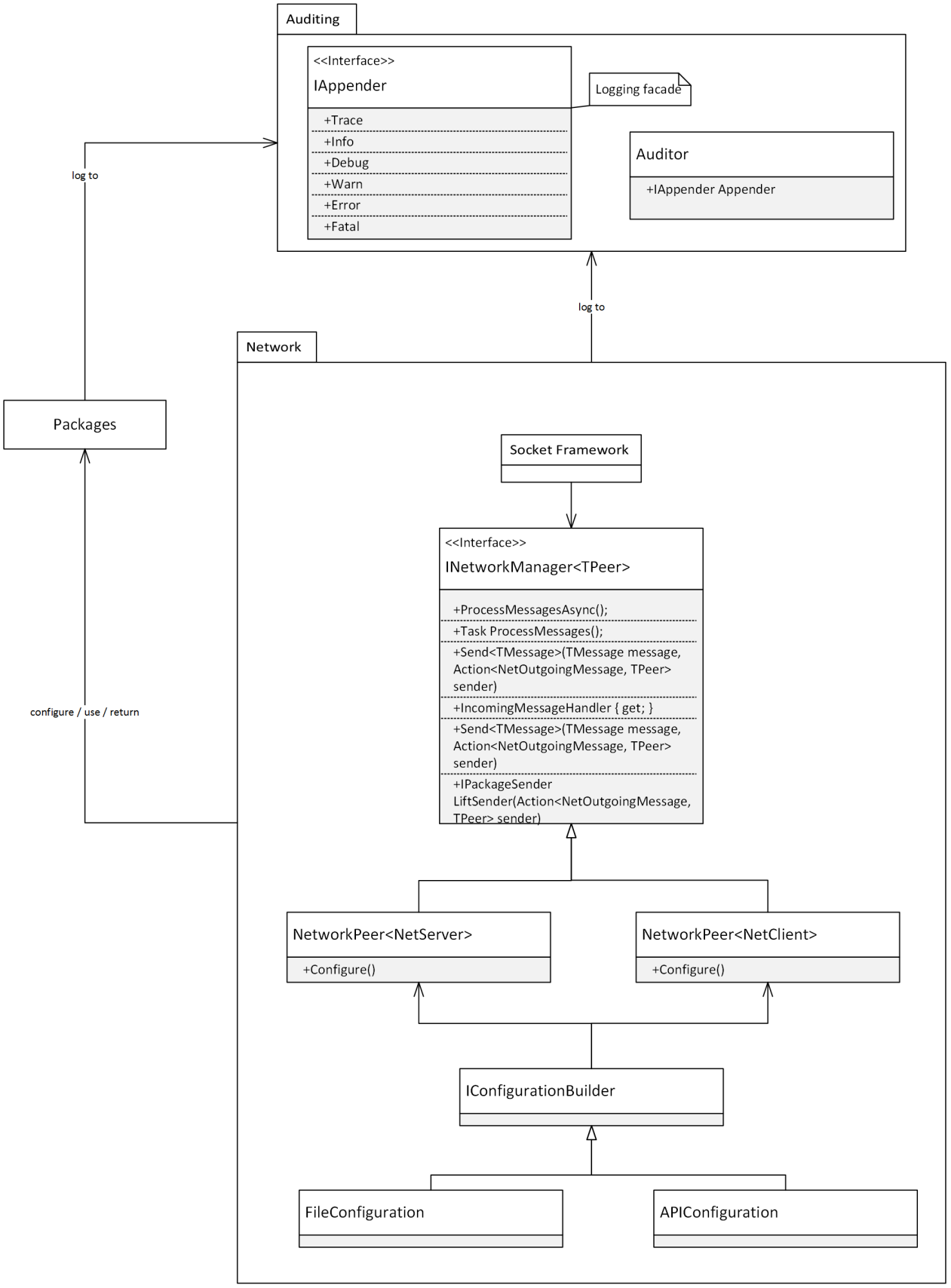
- Κλάσεις που προορίζονται για ανταλλαγή μηνυμάτων

Listing 4.1: Παράδειγμα listener στο initialization

```
[GinetPackage]
 //optional
[PackageSerializer(typeof(MyCustomSerializer))]
public class MyPackage
{
    public string Message { get; set; }
```



Διάγραμμα 4.3: Network Packages Module



Διάγραμμα 4.4: Networking Module

```
}
```

- Ρύθμιση server

Listing 4.2: Παράδειγμα listener στο initialization

```
var server = new NetworkServer("MyServer",
builder =>
{
        //via reflection
        builder.RegisterPackages(
                Assembly.Load("Packages.Assembly.Name"));
        //or manually
        builder.RegisterPackage<MyPackage>();
}
cfg =>
{
        cfg.Port = 1234;
        cfg.ConnectionTimeout = 5.0f;
        cfg.MaxConnections = 10;
        //Additional configuration
}
//optional, logging output, default is the one supplied
out: new ActionAppender(Console.WriteLine)
```

- Καταγραφή κινήσεων

Listing 4.3: Παράδειγμα listener στο initialization

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

- Απάντηση κατά την παραλαβή μηνύματος

Listing 4.4: Παράδειγμα listener στο initialization

```
server.Broadcast<ChatMessage>((msg, im, om) =>
{
        server.Out.Info($"Received␣{msg.Message}");
```

```
        server.SendToAllExcept(om, im.SenderConnection,
            NetDeliveryMethod.ReliableOrdered, channel: 0);
},
packageTransformer: msg =>
msg.Message += "this is broadcasted");



server.BroadcastExceptSender<ChatMessage>((sender, msg) =>
{
        server.Out.Info($"Broadcasting {msg.Message}.
            Received from: {sender}");
});
```

- Κώδικας ο οποίος εκτελείται κατά την παραλαβή συγκεκριμένου τύπου μηνύματος

Listing 4.5: Παράδειγμα listener στο initialization

```
server.IncomingMessageHandler.OnMessage(
        NetIncomingMessageType.ConnectionApproval,
        incomingMsg =>
{
        //Configure connection approval
        var parsedMsg = server
                .ReadAs<ConnectionApprovalMessage>(
                    incomingMsg);
        if(parsedMsg.Password ==
                "my secret and encrypted password")
        {
                incomingMsg.SenderConnection.Approve();
                incomingMsg.SenderConnection.Tag =
                        parsedMsg.Sender;
        }
        else
        {
```

```
                incomingMsg.SenderConnection.Deny();
        }
});
```

- Εκτελέσιμος κώδικας κατά την παραλαβή συγκεκριμένου μηνύματος-κλάσης

Listing 4.6: Παράδειγμα listener στο initialization

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

- Αποστολή μηνύματος-κλάσης

Listing 4.7: Παράδειγμα listener στο initialization

```
server.Send(
        new MyPackage { Message = "Hello" },
(outgoingMessage, peer) =>
        peer.SendMessage(outgoingMessage,
                NetDeliveryMethod.ReliableOrdered);
```

- Lift αποστολής μηνύματος

Listing 4.8: Παράδειγμα listener στο initialization

```
var packageSender = client.LiftSender((msg, peer) =>
        peer.SendMessage(msg,
                NetDeliveryMethod.ReliableOrdered));

packageSender.Send(new MyPackage { Message = "Hello" });
```

- Διαγραφή handler

Listing 4.9: Παράδειγμα listener στο initialization

```
var handlerDisposable = server.IncomingMessageHandler
        .OnPackage<MyPackage>((msg, sender) => {}));
```

```
handlerDisposable.Dispose();
```

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

4.5 Testing

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

Listing 4.10: Incoming Package Handler

```
[TestMethod]
public void Incoming_Package_GetsHandled()
{
        //Arrange
        var client = new NetworkCliient(
                builder =>
                        builder.Register<MyPackage>(),
                cfg=> {});

        var mockedHandler =
                new Mock<Func<MyPackage, NetIncomingMessage>>()
                    ;
        mockedHandler.Setup( x =>
                x(It.IsAny<MyPackage>()));

        client
        .IncomingMessageHandler
```

```
            .OnPackage<MyPackage>(mockedHandler);

            //Act
            client.IncomingMessageHandler.SimulateReceive(
                    new MyPackage());
            client.ProcessMessages().RunSynchronously();

            //Assert
            mockedHandler.Verify(x=>x(), Times.Once()));
}
```

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

4.6 Ανάλυση των επιδόσεων

Benchmarkings here

Πίνακας 4.1: Network Benchmarkings

Κίνητρα	Παραδείγματα ευρημάτων	Αριθμός μελετών
Ταύτιση με το έργο	Ξεκάθαροι στόχοι	20
Καλό management	Ομαδικότητα	16
Συμμετοχή υπαλλήλων	Συμμετοχή στις αποφάσεις	16
Προοπτικές εξέλιξης	Προοπτικές προαγωγής	15
Ποικιλία στην εργασία	Καλή χρήση ικανοτήτων	14
Αίσθηση του να ανήκεις κάπου	Υποστηρικτικές σχέσεις	14
Αμοιβές και κίνητρα	Αυξημένος μισθός	14

4.7 Επεκτασιμότητα

Lobbies and matchmakings

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.. 21

fps Frame rate, also known as frame frequency, is the frequency (rate) at which an imaging device displays consecutive images called frames. The term applies equally to film and video cameras, computer graphics, and motion capture systems. Frame rate is expressed in frames per second (FPS).. 23

gui Γραφικό περιβάλλον χρήστη ή γραφική διασύνδεση/διεπαφή χρήστη (αγγλικά: Graphical User Interface, GUI) καλείται στην πληροφορική ένα σύνολο γραφικών στοιχείων, τα οποία εμφανίζονται στην οθόνη κάποιας ψηφιακής συσκευής (π.χ. Η/Υ) και χρησιμοποιούνται για την αλληλεπίδραση του χρήστη με τη συσκευή αυτή. Παρέχουν στον τελευταίο, μέσω γραφικών, ενδείξεις και εργαλεία προκειμένου αυτός να φέρει εις πέρας κάποιες επιθυμητές λειτουργίες. Για τον λόγο αυτό δέχονται και είσοδο από τον χρήστη και αντιδρούν ανάλογα στα συμβάντα που αυτός προκαλεί με τη βοήθεια κάποιας συσκευής εισόδου (π.χ. πληκτρολόγιο, ποντίκι).. 24

Βιβλιογραφία

Tulach, Jaroslav (2008). *Practical API Design: Confessions of a Java Framework Architect*. Apress.

Η εργασία αυτή στοιχειοθετήθηκε με το πρόγραμμα XeLaTeX. Για τη στοιχειοθέτηση της βιβλιογραφίας χρησιμοποιήθηκε το πρόγραμμα `biber` και `biblatex`. Οι γραμματοσειρές που χρησιμοποιήθηκαν είναι οι Times New Roman και `Courier New`.