ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΕΛΕΤΗ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΕΤΕΡΟΓΕΝΟΥΣ ΣΥΣΤΗΜΑΤΟΣ ΨΥΧΑΓΩΓΙΑΣ ΜΕ ΧΡΗΣΗ ΤΕΧΝΟΛΟΓΙΩΝ HTML5

ΜΠΑΛΤΑΣ Ε. ΑΛΕΞΑΝΔΡΟΣ

ΑΜ. 4384

ΕΠΙΒΛΕΠΩΝ: ΚΑΘΗΓΗΤΗΣ ΠΑΥΛΟΣ ΣΠΥΡΑΚΗΣ
ΣΥΝΕΠΙΒΛΕΠΩΝ: ΔΡ. ΙΩΑΝΝΗΣ ΧΑΤΖΗΓΙΑΝΝΑΚΗΣ

ΠΑΤΡΑ, ΙΟΥΛΙΟΣ 2013

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κ. Παύλο Σπυράκη, Καθηγητή του Τμήματος Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής και επιβλέποντα της διπλωματικής μου, για την εμπιστοσύνη που μου έδειξε δίνοντας μου την ευκαιρία να ασχοληθώ με το αντικείμενο αυτής της εργασίας.

Επίσης, ένα θερμό ευχαριστώ στον Δρ. Ιωάννη Χατζιγιαννάκη, συνεπιβλέποντα της διπλωματικής μου, για την εμπιστοσύνη, τη βοήθεια και την καθοδήγηση του σε όλη τη διάρκεια εκπόνησης αυτής της εργασίας. Ο τρόπος σκέψης που μου μετέδωσε, και το κλίμα συνεργασίας που δημιούργησε έπαιξαν καθοριστικό ρόλο στα αποτελέσματα της.

Θέλω επίσης να ευχαριστήσω τους Ορέστη Ακριβόπουλο και Μάριο Λογαρά για την πολύτιμη βοήθεια τους και τη συνεργασία που είχαμε κατά τη διάρκεια εκπόνησης αυτής της εργασίας.

Τέλος, ένα θερμό ευχαριστώ στους γονείς μου Μανώλη & Ευτυχία και στα αδέρφια μου Άκη & Σωκράτη για την στήριξη τους σε όλα τα βήματα της ζωής μου με κάθε τρόπο, καθώς και σε όλους τους φίλους μου που ήταν κοντά μου όλα αυτά τα χρόνια.

Μπαλτάς Αλέξανδρος

Πάτρα, Ιούλιος 2013

ΠΕΡΙΕΧΟΜΕΝΑ

ΛΙΣΤΑ ΕΙΚΟΝΩΝ ΚΑΙ ΣΧΗΜΑΤΩΝ

1 Εισαγωγή

1.1 Κίνητρο και Σημασία

Τα τελευταία χρόνια η ανάπτυξη και διάδοση των νέων τεχνολογιών στο πεδίο του Παγκόσμιου Ιστού είναι ραγδαία. Ο τρόπος με τον οποίο αντιμετωπίζουμε το Internet αλλάζει, αφού η δυνατότητα συνδεσιμότητας είναι πια σχεδόν αυτονόητη. Με την έλευση της HTML5 άνοιξαν νέοι δρόμοι στον τρόπο με τον οποίο παρουσιάζεται το περιεχόμενο στον Παγκόσμιο Ιστο. Μαζί της δημιουργήθηκαν πολλές τεχνολογίες που έχουν να κάνουν με διασύνδεση και επικοινωνία απομακρυσμένων συστημάτων.

Παράλληλα, τα ασύρματα δίκτυα αισθητήρων είναι ένα πεδίο που ολοένα και παρουσιάζει μεγαλύτερο ερευνητικό ενδιαφέρον. Ένας από αυτούς τους λόγους είναι και το παραπάνω γεγονός, η ευκολία δηλαδή με την οποία κανείς μπορεί να συνδεθεί στο Internet. Έτσι γεννήθηκε και η ιδέα του Internet of Things, ενός Internet δηλαδή που θα δουλεύει για εμάς σε όλα τα αντικείμενα της καθημερινής μας ζωής, χρησιμοποιώντας ασύρματα δίκτυα αισθητήρων.

Τέλος, η χρήση έξυπνων προσωπικών συσκευών, όπως smartphones και tablets στην καθημερινή μας ζωή συνεχώς αυξάνεται. Ενδεικτικό είναι πως υπολογίζεται ότι τον Ιούνιο του 2013 το 61% των Αμερικανών πολιτών είναι χρήστες smartphone.

Όλα αυτά λοιπόν ευνοούν στη δημιουργία ετερογενών συστημάτων, συστημάτων δηλαδή που θα χρησιμοποιούν περισσότερα από ένα είδη συσκευών, τα οποία θα συνδέονται μεταξύ τους μέσω του Παγκόσμιου Ιστού.

1.2 Στόχοι της διπλωματικής εργασίας

Η παρούσα διπλωματική εργασία έχει ως στόχο τη μελέτη, την σχεδίαση και την υλοποίηση μιας πλατφόρμας παιχνιδιών διάχυτου υπολογισμού, η οποία θα χρησιμοποιεί τις δυνατότητες των ασύρματων δικτύων αισθητήρων, σε συνδυασμό με τις δυνατότητες των smartphones και τις τεχνολογίες του Future Internet. Βάση αυτής της πλατφόρμας θα αποτελεί η ήδη υλοποιημένη πλατφόρμα FunInNumbers, με τη διαφορά πως η νέα αρχιτεκτονική θα δίνει περισσότερη έμφαση στις τεχνολογίες Παγκόσμιου Ιστου.

1.3 Δομή της διπλωματικής εργασίας

Στο κεφάλαιο 2 παρουσιάζονται τεχνολογίες, εργαλεία και τεχνικές που χρησιμοποιήθηκαν για την σχεδίαση και την υλοποίηση της πλατφόρμας.

Στο κεφάλαιο 3 παρουσιάζεται η πλατφόρμα ανάπτυξης διαδραστικών παιχνιδιών FunInNumbers, πάνω στην οποία βασίζεται και η αρχιτεκτονική που σχεδιάστηκε στα πλαίσια αυτής της εργασίας.

Στο κεφάλαιο 4 παρουσιάζεται η προτεινόμενη αρχιτεκτονική του συστήματος και αναλύονται τα διάφορα επίπεδα της.

Στο κεφάλαιο 5 παρουσιάζονται ζητήματα υλοποίησης που προέκυψαν κατά τη διάρκεια της εκπόνησης αυτής της εργασίας

Στο κεφάλαιο 6 παρουσιάζετα η υλοποίηση 2 παιχνιδιών σύμφωνα με την αρχιτεκτονική που σχεδιάστηκε.

Στο κεφάλαιο 7 αναφέρονται μελλοντικές κατευθύνσεις για την πλατφόρμα.

2 Τεχνολογίες - Εργαλεία

2.1 HTML5

Η HTML5 **[1]** είναι μία γλώσσα περιγραφής και παρουσίασης περιεχομένου στον Παγκόσμιο Ιστό. Αποτελεί την πέμπτη έκδοση της HTML, η οποία δημιουργήθηκε το 1990. Στόχος της HTML5 είναι να βελτιώσει τη γλώσσα και να υποστηρίξει τα σύγχρονα πολυμέσα (ήχο, εικόνα, video), ενώ παραμένει ευανάγνωστη για τους ανθρώπους και αναπαρίσταται με συνέπεια από τους υπολογιστές.

Εικόνα 2.1 Το λογότυπο της HTML5

2.2 Google Web Toolkit

Το Google Web Toolkit **[2]** είναι ένα σύνολο από εργαλεία για την ανάπτυξη πολύπλοκων εφαρμογών ιστού. Ο λόγος για τον οποίο αναπτύχθηκε το συγκεκριμένο σύνολο εργαλείων ήταν η ανάγκη για την γρήγορη ανάπτυξη σύνθετων διαδικτυακών εφαρμογών που δεν θα απαιτούσαν από τον προγραμματιστή να έχει εξαιρετικές γνώσεις σε τεχνολογίες όπως JavaScript και AJAX **[3]**. Ο προγραμματιστής μπορεί να χρησιμοποιεί ένα διαγλωσσικό

μεταφραστή από Java σε JavaScript με αποτέλεσμα να προγραμματίζει στην γλώσσα Java και μέσω του μεταφραστή να παράγει λειτουργικό, συμπαγή και βελτιστοποιημένο κώδικα JavaScript.

Ένα από τα προβλήματα ανάπτυξης εφαρμογών JavaScript είναι η ανύπαρκτη (χωρίς βοήθεια ειδικού λογισμικού) αποσφαλμάτωση κώδικα. Με τη χρήση αυτού το μεταφραστή μπορούν να βρεθούν εύκολα σφάλματα στον κώδικα και να αποφευχθεί το πρόβλημα της αποσφαλμάτωσης της JavaScript στης εφαρμογές Παγκόσμιου Ιστού.

Ένα άλλο πλεονέκτημα που παρέχει η Java στο GWT είναι η εικονική μηχανή της Java. Το GWT μπορεί να χρησιμοποιηθεί σε δύο λειτουργίες. Η πρώτη ονομάζεται λειτουργία παραγωγής (Production Mode) η οποία χρησιμοποιεί τον μεταφραστή που περιγράφεται παραπάνω. Η άλλη λειτουργία ονομάζεται λειτουργία ανάπτυξης (Development Mode) και χρησιμοποιεί την εικονική μηχανή της Java ώστε ο κώδικας να εκτελεστεί στη μηχανή αυτή χωρίς να μεταφράζεται σε JavaScript. Αυτό επιτρέπει στην εύκολη και γρήγορη δοκιμή της υπό ανάπτυξης εφαρμογή σε ένα σύγχρονο περιηγητή ιστού απλά με τη χρήση ενός προσθέτου (plugin).

Πολύ σημαντικό χαρακτηριστικό του Google Web Toolkit είναι η δυνατότητα που δίνει στον προγραμματιστεί να αναμίξει Java με Javascript. Στην περίπτωση που δεν καλύπτεται από το API που παρέχεται απο το GWT, ο χρήστης μπορεί μέσω του JSNI (JavaScript Native Interface) να ορίσει Java μεθόδους που εκτελούν Javascript κώδικα.

Εικόνα 2.2 Το λογότυπο του Google Web Toolkit

2.3 Google Protocol Buffers

2.3.1 Serialization

Η διαδικασία του Serialization λύνει το πρόβλημα της διατήρησης των δεδομένων επιτρέποντας την αποθήκευση και την ανταλλαγή αντικειμένων ως μια ακολουθία από bits. Με τον όρο serialization αναφερόμαστε στην διαδικασία μετατροπής ενός αντικειμένου σε μια ακολουθία απο bits με σκοπό να διατηρηθούν σε κάποιο μέσο αποθήκευσης ή να αποσταλούν μέσω κάποιου δικτύου. Η ακολουθία αυτή των bits όταν ξαναδιαβάζεται από το μέσο αποθήκευσης ή από τον δέκτη, μπορεί να δημιουργήσει ένα ακριβές αντίγραφο του αρχικού αντικειμένου. Για ιδιαίτερα περίπλοκα αντικείμενα τα οποία περιέχουν αναφορές σε άλλα αντικείμενα η διαδικασία αυτή δεν μπορεί να γίνει απευθείας.

Η διαδικασία του serialization καλείται επίσης deflating ή marshalling ενός αντικειμένου. Η αντίθετη διαδικασία, δηλαδή η δημιουργία μια συγκεκριμένης δομής δεδομένων από μία ακολουθία από bits, καλείται deserialization (είναι επίσης γνωστή και ως inflating ή unmarshalling).

Με το serialization δεδομένων:

- Γίνεται δυνατή η κλήση απομακρυσμένων μεθόδων (Remote Procedure Calls).
- Γίνεται εύκολος και αποδοτικός διαμοιρασμός δεδομένων ανάμεσα σε διαφορετικές εφαρμογές.
- Προσφέρονται αποδοτικές μέθοδοι αναγνώρισης τροποποιήσεων σε δεδομένα με την πάροδο του χρόνο, αφού είναι ευκολότερο να ανιχνευθούν τροποποιήσεις σε ακολουθίες bit.

2.3.2 Protocol Buffers

Τα Protocol Buffers **[4]** (Protobufs) είναι μια μέθοδος serialization δομημένων δεδομένων. Η μέθοδος αποτελείται από μια γλώσσα με τη χρήση της οποίας περιγράφεται η δομή των δεδομένων, και ένα πρόγραμμα που δημιουργεί το stream των bytes που προκύπτει από αυτή την περιγραφεί και αντιπροσωπεύει τα δεδομένα. Η Google ανέπτυξε αυτή τη μέθοδο για να χρησιμοποιηθεί σε δικές της εφαρμογές, διαθέτει compilers για C++, Java και Python, και τη διαθέτει στο κοινό με άδεια χρήσης δωρεάν ανοιχτού λογισμικού (free open source software). Έτσι έχουν δημιουργηθεί και υλοποιήσεις σε αρκετές ακόμα

γλώσσες (πχ. Objective-C, Ruby). Οι στόχοι των Protocol Buffers ήταν η απλότητα υλοποίησης και η απόδοση. Συγκεκριμένα σχεδιάστηκαν για να είναι πιο αποδοτικά σε χρόνο και σε όγκο από αντίστοιχα δεδομένα σε XML.

Η Google χρησιμοποιεί τα Protobufs για την αποθήκευση και την αποστολή πολλών ειδών δομημένης πληροφορίας. Μάλιστα, αποτελούν τη βάση ενός εξειδικευμένου συστήματος RPC (Remote Procedure Call) που χρησιμοποιείται σχεδόν για όλες τις ανταλλαγές δεδομένων των συστημάτων της Google.

Για να χρησιμοποιηθούν τα protobufs ο χρήστης περιγράφει τη δομή δεδομένων που χρειάζεται (η οποία καλείται message) σε ένα αρχείο περιγραφής (.proto) και το μεταφράζει με το εργαλείο protoc. Η μετάφραση του αρχείου δημιουργεί κώδικα ο οποίος μπορεί να χρησιμοποιηθεί από αποστολείς ή αποδέκτες αυτής της δομής δεδομένων. Για παράδειγμα, από τη μετάφραση ενός αρχείου ex.proto θα προκύψουν τα αρχεία ex.pb.cc και ex.pb.h στα οποία θα ορίζουν κλάσεις c++ για κάθε message που ορίζεται στο ex.proto.

Ένα παράδειγμα ενός αρχείου περιγραφής φαίνεται παρακάτω:

```
message Rect {
  required int32 x_length = 10;
  required int32 y_length = 20;
  optional string label;
```

Εικόνα 2.3 Παράδειγμα περιγραφής δεδομένων σε αρχείο .proto

2.4 Processing.js

Η Processing.js **[5]** είναι η γλώσσα που αποτελεί την μετατροπή της Processing που αναλύεται παρακάτω, σε Javascript. Μέσω της Processing.js οι περιηγητές ιστού μπορούν να απεικονίζουν κινούμενα σχέδια, παιχνίδια και γενικά γραφικά πλούσιο περιεχόμενο χωρίς να χρειάζεται Java Applet ή Flash Plugin.

Η Processing.js χρησιμοποιεί Javascript για σχεδιάσει γραφικά δύο και τριών διαστάσεων στον canvas της HTML5, και υποστηρίζεται από όλους τους

μοντέρνους περιηγητές ιστού που υποστηρίζουν αυτό το στοιχείο (τελευταίες εκδόσεις των Mozilla Firefox, Opera, Apple Safari, Google Chrome, Internet Explorer 9).

Η ανάπτυξη της ξεκίνησε από τον John Resig, και μετά συνεχίστηκε από φοιτητές του Seneca College μετά το 2008 οι οποίοι διόρθωσαν περίπου 900 σφάλματα, και την ολοκλήρωσαν, βγάζοντας 12 εκδόσεις. Όλη η διαδικασία αποτελούσε συνεργασία του Mozilla Foundation και του Seneca College.

Η Processing **[6]** είναι μια γλώσσα προγραμματισμού ανοιχτού κώδικα, που συνοδεύεται από ολοκληρωμένο περιβάλλον προγραμματισμού (integrated development environment - IDE) που χρησιμοποιείται για εφαρμογές παραγωγής εικόνας και κινουμένων σχεδίων. Βασίζεται στη Java, αλλά χρησιμοποιεί απλοποιημένη σύνταξη . Στην πράξη κάθε πρόγραμμα αποτελεί υποκλάση της Java κλάσης PApplet και διαθέτει 2 βασικές μεθόδους:
• **setup()**: Εκτελείται μια φορά, στην αρχή του προγράμματος
• **draw()**: Εκτελείται συνέχεια, μετά το τέλος της setup().

Εικόνα 2.4 Το λογότυπο της Processing.js

2.5 Websockets

Τα WebSockets **[7]** είναι ένα πρωτόκολλο που επιτρέπει αμφίδρομη επικοινωνία πάνω από μία TCP σύνδεση, και έχει προτυποποιηθεί από την IETF ως RFC 6455. Αποτελεί ένα ανεξάρτητο, βασισμένο στο TCP, πρωτόκολλο που η μόνη του σύνδεση με το HTTP είναι πως οι διακομιστές ιστού διαχειρίζονται το handshake του πρωτοκόλλου σαν αίτημα HTTP

Upgrade. Η TCP θύρα που χρησιμοποιείται από τα websockets είναι η 80 που είναι η καθιερωμένη θύρα του HTTP πρωτοκόλλου.

Για να εγκατασταθεί μια σύνδεση πρέπει ο περιηγητής ιστού να στείλει ένα WebSocket handshake αίτημα και ο web server να απαντήσει με μία WebSocket handshake απάντηση, όπως φαίνεται στο ακόλουθο παράδειγμα:

```
GET /mychat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

Εικόνα 2.5 WebSocket Αίτημα

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

Εικόνα 2.6 WebSocket Απάντηση

Παρόλο που αρχικά σχεδιάστηκε για χρήση από διακομιστές ιστού, και περιηγητές ιστού αντίστοιχα, μπορεί να χρησιμοποιηθεί σε οποιαδήποτε εφαρμογή που ακολουθεί το μοντέλο πελάτη-διακομιστή. Έτσι, προσφέρεται εύκολη, αμφίδρομη διαρκής επικοινωνία, για όσο η σύνδεση παραμένει ανοιχτή.

Πολύ σημαντικό πλεονέκτημα των WebSockets είναι η ευκολία που προσφέρουν στον προγραμματιστή στην ανάπτυξη ενός προγράμματος

βασισμένου σε αυτά. Πρακτικά χρειάζεται να υλοποιηθούν 4 βασικές συναρτήσεις. Σε διάφορες υλοποιήσεις υπάρχουν και πιο εξειδικευμένες συναρτήσεις αλλά οι 4 βασικές είναι οι εξής:

- **onOpen()**: Εκτελείται όταν το WebSocket ανοίγει.
- **onClose()**: Εκτελείται όταν το WebSocket κλείνει.
- **onMessage()**: Εκτελείται όταν το WebSocket δεχθεί μήνυμα. Συνήθως υλοποιούνται 2 συναρτήσεις, η **onTextMessage()** που εκτελείται όταν το WebSocket δεχθεί μήνυμα σε μορφή String, και η **onBinaryMessage()** η οποία εκτελείται όταν το WebSocket δεχθεί μήνυμα που αποτελείται από Raw Data.
- **onError()**: Εκτελείται όταν συμβεί κάποιο σφάλμα.

2.6 SunSPOT

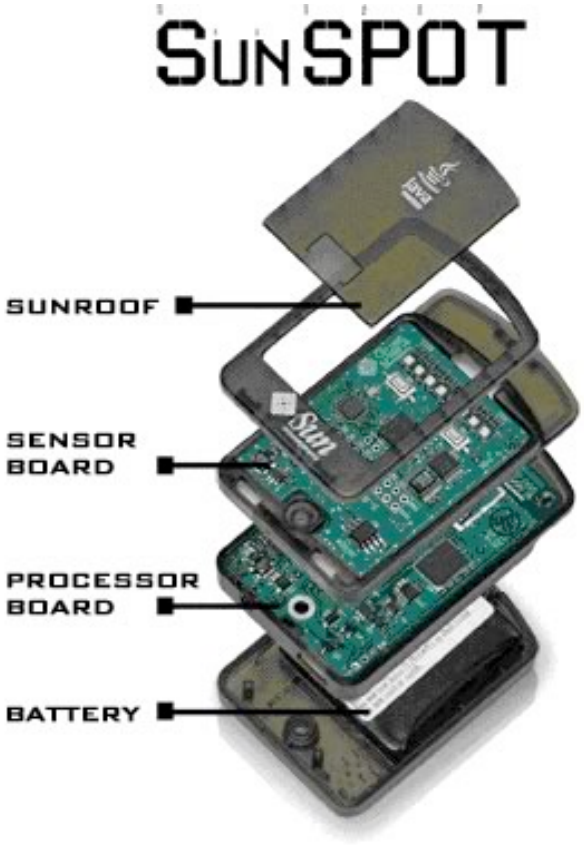
Το SunSPOT **[8]** είναι μια μικρή, ασύρματη πειραματική πλατφόρμα που εντάσσεται στην κατηγορία των αισθητήρων και προγραμματίζεται εξολοκλήρου σε Java γεγονός. Η συσκευή περιλαμβάνει ένα πλήθος αισθητήρων καθώς και παρέχει τη δυνατότητα διασύνδεσης εξωτερικών συσκευών και επιπλέον πλακετών.
Η συσκευή αποτελείται από διάφορα επίπεδα υλικού δύο εκ των οποίων και τα πιο σημαντικά: η βασική πλακέτα (eSpot board) και η πλακέτα των αισθητήρων (eDemo board).

Εικόνα 2.7 Ένας αισθητήρας Sun SPOT

Σχήμα 3.2: Τα διαφορετικά επίπεδα υλικού που συνθέτουν ένα Sun SPOT. Από κάτω προς τα πάνω: μπαταρία, βασικά πλακέτα επεξεργαστή, πλακέτα αισθητήρων, καπάκι.
Το βασικό developement kit των SunSPOT περιλαμβάνει δύο ειδών συσκευές:

- **Basestation** Το Basestation αποτελείται μόνο από την βασική πλακέτα eSpot χωρίς μπαταρία ή επιπλέον πλακέτα. Τροφοδοτείται μέσω της USB σύνδεσης σε υπολογιστή ο οποίος χρησιμοποιείται και σαν host workstastion για τη συσκευή. Η κυριότερη λειτουργία του basestation είναι αυτή του gateway μεταξύ των Sun SPOT και γενικότερα συσκευών συμβατών με το IEEE 802.15.4 και του workstastion.

- **eSPOT** Η μονάδα περιλαμβάνει επιπλέον μια επαναφορτιζόμενη μπαταρία και τη δευτερεύουσα πλακέτα eDemo board.



Εικόνα 2.8 Σχηματική αναπαράσταση των μερών ενός Sun SPOT και της μεταξύ τους διασύνδεσης.

Η πλακέτα **eSpot main board** αποτελείται από:

• Κεντρικό επεξεργαστή
• Μνήμη
• Κύκλωμα διαχείρισης τροφοδοσίας
• Πομποδέκτη 802.15.4 με ενσωματωμένη κεραία
• Σύνδεση μπαταρίας
• Σύνδεση δευτερεύουσας πλακέτας.

Ο **κεντρικός επεξεργαστής** είναι ένας Atmel ARM920T ARM Thumb processor ενσωματωμένος σε ένα System On Chip (SOC) κύκλωμα, το AT91RM9200. Η μέγιστη συχνότητα λειτουργίας του φτάνει τα 180 Mhz. Τα περιεχόμενα της μνήμης διατηρούνται εφόσον υπάρχει τροφοδοσία. Η συσκευή τροφοδοτείται είτε από την μπαταρία είτε από μια εξωτερική πηγή μέσω της σύνδεσης USB. Το κύκλωμα ελέγχου τροφοδοσίας είναι υπεύθυνο για την φόρτιση της μπαταρίας, ρυθμίζει την τροφοδοσία των δευτερευουσών και της κεντρικής πλακέτας, παρέχει ρεύμα κατά την κατάσταση deep-sleep, διατηρεί των μετρητή του εσωτερικού ρολογιού 64 bit ενώ ελέγχει και παρακολουθεί τους διακόπτες και τα αντίστοιχα LED. Η ασύρματη επικοινωνία του Sun SPOT είναι δυνατή μέσω του πρωτοκόλλου IEEE 802.15.4.

Η πλακέτα των αισθητήρων eDemo Board είναι εξοπλισμένη με τα εξής:

• Atmega88 επεξεργαστή
• μνήμη flash
• αισθητήρα επιτάχυνσης τριών αξόνων με δύο ρυθμίσεις επιλογής εύρους (2G ή 6G)
• αισθητήρα θερμότητας αισθητήρα φωτός
• 8 RGB LEDs
• 6 αναλογικές εισόδους
• 2 διακόπτες
• 5 pins γενικής χρήσης
• 4 pins εξόδου υψηλού δυναμικού

Τα Sun SPOT χρησιμοποιούν μια έκδοση της Java με τις δυνατότητες της Java Microedition και λέγεται Squawk **[9]**. Το Squawk υποστηρίζει τα πακέτα CLDC 1.1 και MIDP 1.0 και παρέχει τα βασικά ενός Λειτουργικού Συστήματος. Η Virtual Machine εκτελείται απευθείας από την μνήμη Flash. Ο σχεδιασμός

της είναι ιδανικός για εκτέλεση σε ενσωματωμένα συστήματα και μικρές συσκευές. Σε αντίθεση με τις περισσότερες Virtual Machines για Java ο πυρήνας των οποίων είναι γραμμένος σε C/C++, ο πυρήνας της Squawk είναι γραμμένος σε Java.

Για τον προγραμματισμό των Sun SPOT χρησιμοποιείται το Sun SPOT SDK που περιλαμβάνει ένα σύνολο βιβλιοθηκών για το χειρισμό των συστημάτων της συσκευής. Η έκδοση 5.0 του SDK περιέχει τρία ξεχωριστά πακέτα βιβλιοθηκών:
1. SPOT and Sensorboard libraries - Μεταξύ άλλων δίνει πρόσβαση στη μνήμη, στις περιφερειακές συσκευές του main board, στο wireless radio, στα υποστηριζόμενα routing πρωτόκολλα, επιτρέπει τη χρήση του sensorboard καθώς και παρέχει πακέτα για την εκτέλεση δοκιμαστικών εφαρμογών.
2. SPOT Generic Connection Framework - Δίνει πρόσβαση σε όλους τους συσκευές εισόδου/εξόδου του Sun SPOT.
3. Squawk Java ME library - Περιέχει όλες της βιβλιοθήκες για τον έλεγχο της Squawk VM.

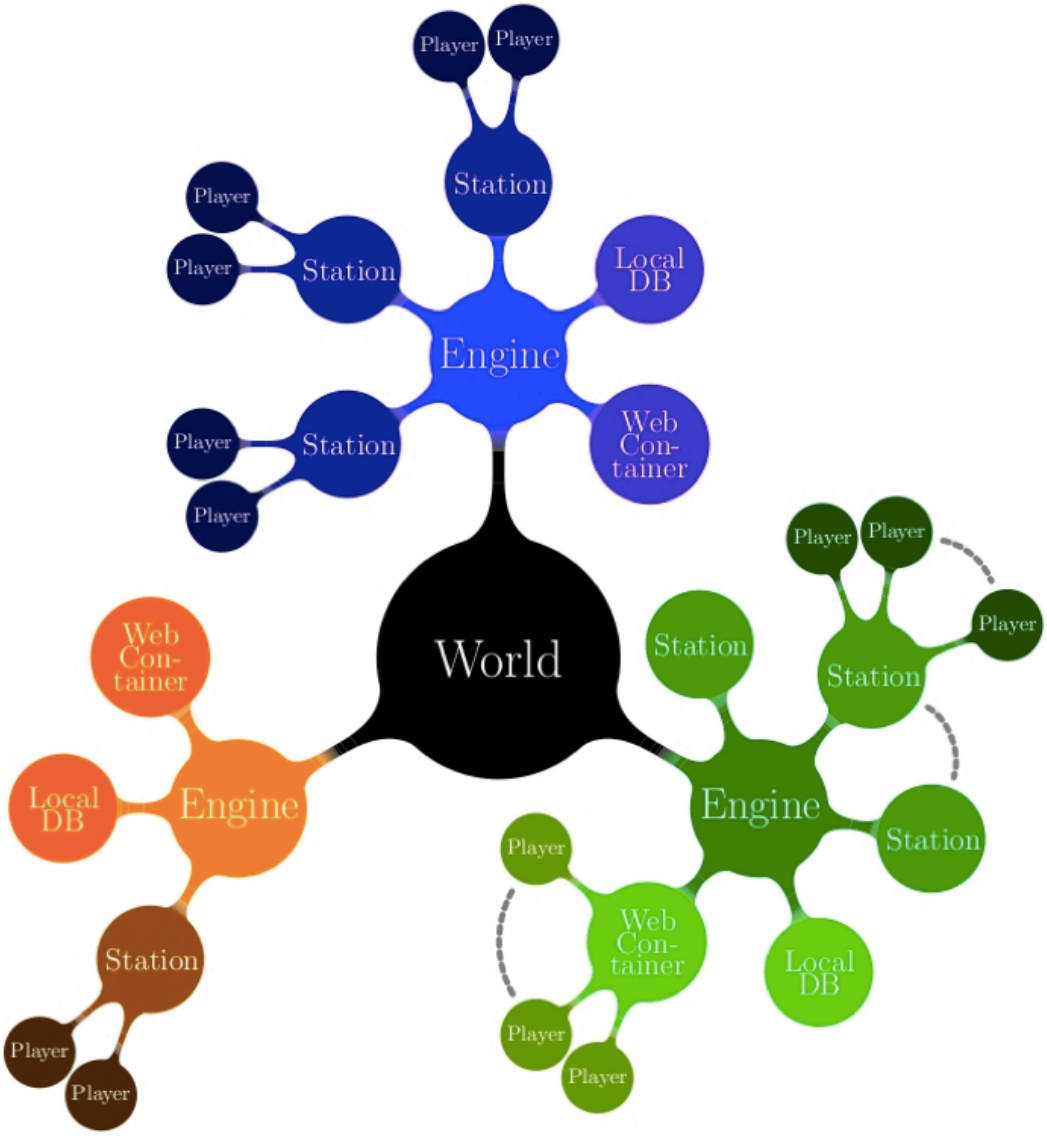
3 Πλατφόρμα Fun In Numbers

3.1 Εισαγωγή

Η πλατφόρμα Fun In Numbers **[10]** είναι μια πλατφόρμα ανάπτυξης διαδραστικών παιχνιδιών με χρήση ασύρματων δικτύων αισθητήρων.

3.2 Αρχιτεκτονική

Στην ενότητα αυτή αναλύεται η αρχιτεκτονική της πλατφόρμας Fun In Numbers **[11]** που περιλαμβάνει κινητά δίκτυα αισθητήρων και υπολογιστικά συστήματα τα οποία είναι συνδεδεμένα μεταξύ τους καθώς επίσης και με το διαδίκτυο. Στο σχήμα 3.1 παρουσιάζεται αρχιτεκτονική του συστήματος.



Εικόνα 3.1 Αρχιτεκτονική του Συστήματος

Η αρχιτεκτονική του συστήματος βασίζεται σε μια ιεραρχία επιπέδων προσφέροντας επεκτασιμότητα καθώς επίσης και την χρήση διαφορετικών συσκευών (heterogeneity). Έχουν αναπτυχθεί πρωτόκολλα που προσφέρουν χωρικό προσδιορισμό (location awareness) ασυρμάτων συσκευών σε εσωτερικούς χώρους, εκμεταλλεύονται τις δυνατότητες των αισθητήρων, συντονίζουν βασικές κατανεμημένες λειτουργίες και προσφέρουν επίσης επικοινωνία ανεκτική σε καθυστερήσεις (delay-tolerant communication). Επίσης στατιστικά συλλέγονται από τις κινητές συσκευές των παικτών, επεξεργάζονται και στη συνέχεια αποθηκεύονται σε μία κεντρική βάση

δεδομένων καθιστώντας δυνατή την προβολή τους σε οποιοδήποτε μελλοντικό χρόνο μέσω μίας ιστοσελίδας.

Η συγκεκριμένη αρχιτεκτονική επιτρέπει την ανάπτυξη παιχνιδιών απόκρισης πραγματικού χρόνου, παιχνιδιών που απαιτούν χωρικό προσδιορισμό, παιχνιδιών που εμπλουτίζουν τον πραγματικό χώρο (augmented reality) καθώς επίσης και παιχνιδιών που δεν απαιτούν σύνδεση με το διαδίκτυο. Ο κύριος στόχος ήταν η ανάπτυξη μίας πλατφόρμας λογισμικού η οποία θα επέτρεπε την εύκολη ενσωμάτωση της σε συστήματα διαφορετικού υλικού και γιαυτό το λόγο η γλώσσα που επιλέχθηκε ήταν η Java. Η Java είναι μια ευρέως διαδεδομένη γλώσσα προγραμματισμού η οποία μπορεί χρησιμοποιηθεί σε διάφορες συσκευές, π.χ. κινητές συσκευές, αισθητήρες, καθώς επίσης και σε οποιοδήποτε υπολογιστικό σύστημα. Οι βιβλιοθήκες που προσφέρει επιτρέπουν την διασύνδεση, επικοινωνία των εφαρμογών με βάσεις δεδομένων και την ανταλλαγή πληροφοριών μεταξύ συσκευών μέσω διάφορων καναλιών επικοινωνίας. Επίσης οι εφαρμογές που έχουν αναπτυχθεί σε Java μπορούν να είναι ανεξάρτητες του υλικού των συσκευών στις οποίες θα εκτελούνται. Για την μελέτη της συμπεριφοράς του συστήματος έχουν αναπτυχθεί διάφορα παιχνίδια τα οποία παρουσιάζουν τις δυνατότητες του. Κάθε ένα από αυτά εστιάζει σε διαφορετικά χαρακτηριστικά του συστήματος. Στη συνέχεια αναλύονται όλα τα επίπεδα της αρχιτεκτονικής ξεχωριστά και οι σχεδιαστικές αποφάσεις που ελήφθησαν με σκοπό να λύσουν τις βασικές προκλήσεις ενός τέτοιου ετερογενούς συστήματος.

3.2.1 Guardian

Το επίπεδο του Guardian είναι το χαμηλότερο επίπεδο στην αρχιτεκτονική του συστήματος. Αποτελείται από τις συσκευές που χρησιμοποιούν οι παίκτες κατά την διάρκεια των παιχνιδιών. Οι Guardians είναι το λογισμικό που εκτελείται στις συσκευές των παικτών και χρησιμοποιεί τους αισθητήρες της με σκοπό την δημιουργία μίας διεπαφής για την αλληλεπίδραση με τον χρήστη, την επικοινωνία κτλ. Προσφέρει επίσης ένα πρωτόκολλο για την εύρεση γειτόνων, την επικοινωνία με την υπόλοιπη υποδομή και με τους υπόλοιπους Guardians (echo protocol service). Όταν ανακαλυφθεί ένας νέος Guardian ο παίχτης έχει την δυνατότητα να προβεί σε περαιτέρω ενέργειες, είτε χρησιμοποιώντας τους αισθητήρες είτε τα κουμπιά της συσκευής. Για την παρακολούθηση της εξέλιξης του παιχνιδιού κάθε ενέργεια η οποία σχετίζεται με το παιχνίδι αναπαρίσταται από ένα Event.
Στους Guardians επίσης έχει υλοποιηθεί μια υπηρεσία που προσφέρει την αλληλεπίδραση με άλλους Guardians ακόμη και όταν αυτοί είναι

αποσυνδεδεμένοι από την υπόλοιπη υποδομή του συστήματος για μεγάλα χρονικά διαστήματα. Συγκεκριμένα, όταν δημιουργείται ένα νέο Event, ο Guardian το αποθηκεύει τοπικά στην μνήμη του και μόλις η επικοινωνία με την υπόλοιπη υποδομή είναι δυνατή, όλα τα αποθηκευμένα Events προωθούνται σε ανώτερα επίπεδα της αρχιτεκτονικής (delay tolerant communication service). Επίσης οι Guardians παρέχουν ένα υποσύστημα το οποίο επεξεργάζεται τα δεδομένα που συλλέγονται από τον αισθητήρα επιτάχυνσης και αναγνωρίζει τις κινήσεις που έγιναν με την συσκευή (gestures) και τις αντιστοιχεί σε ενέργειες σχετικές με το παιχνίδι. Οι παίκτες δηλαδή κατά την διάρκεια του παιχνιδιού δεν ενεργούν μόνο με το πάτημα κάποιων κουμπιών της συσκευής αλλά κυρίως κάνοντας κινήσεις με την ίδια την συσκευή.

3.2.2 Station

Το επίπεδο των Stations είναι υπεύθυνο για της επικοινωνία και την διασύνδεση των συσκευών που έχουν οι παίκτες με την υπόλοιπη υποδομή του δικτύου του συστήματος και είναι πολύ σημαντικό αλλά όχι απαραίτητο για όλα τα παιχνίδια που έχουν υλοποιηθεί. Προσφέρει υπηρεσίες context awareness και μέσω των Stations τα δεδομένα μεταφέρονται από και προς τα υψηλότερα επίπεδα της αρχιτεκτονικής με σκοπό τον συντονισμό των παιχνιδιών και την μόνιμη αποθήκευση των δεδομένων. Κάθε Station ελέγχει μία συγκεκριμένη φυσική περιοχή.

Τα Stations επικοινωνούν με τις συσκευές των χρηστών είτε μέσω τοπικών adhoc δικτύων είτε μέσω personal area non-IP δικτύου και λειτουργούν ως προεπιλεγμένες πύλες, επιτρέποντας στην ουσία την επικοινωνία των Guardians με το αμέσως υψηλότερο επίπεδο, που είναι το Game Engine. Πολλαπλά Stations μπορούν να είναι συνδεδεμένα σε ένα Game Engine αυξάνοντας με αυτόν τον τρόπο την περιοχή στην οποία μπορούν να εξελιχθούν τα παιχνίδια. Κατά την αρχικοποίηση των παιχνιδιών τα Stations επικοινωνούν με το Game Engine και ανακτούν δεδομένα όπως οι παίκτες που είναι εγγεγραμμένοι για το συγκεκριμένο παιχνίδι το οποίο εξελίσσεται, τις συσχετίσεις ανάμεσα στις οντότητες του διαγράμματος συσχέτισης (ER) του σχήματος 3.2 και τους εγγεγραμμένους Guardians. Τα Stations είναι υπεύθυνα για την αρχικοποίηση των Guardians και για την προώθηση όλων των δεδομένων που δημιουργούνται κατά την εξέλιξη ενός παιχνιδιού στο Game Engine.
Υπάρχει επίσης ακόμη μια επιλογή για τον τρόπο λειτουργίας των Stations κατά την διάρκεια ενός παιχνιδιού που δεν σχετίζεται με τα παραπάνω. Σε

αυτήν την περίπτωση ονομάζονται mobile Stations και λειτουργούν με διαφορετικό τρόπο. Δεν προσφέρουν επικοινωνία με τα υψηλότερα επίπεδα της αρχιτεκτονίκης παρά μόνο κάλυψη μεγαλύτερης περιοχής και location-aware υπηρεσίες. Ενημερώνουν τους Guardians αποστέλλοντας Events για την mobile λειτουργία τους και αυτοί με την σειρά τους είναι υπεύθυνοι μέσω των σταθερών Stations να ενημερώσουν το Game Engine.

3.2.3 Engine

Κάθε παιχνίδι που αρχικοποιείται και ετοιμάζεται να ξεκινήσει ανατίθεται σε ένα συγκεκριμένο Game Engine από το οποίο συντονίζεται. Το Engine είναι υπεύθυνο στην ουσία για την επιβολή των κανόνων κάθε παιχνιδιού. Με σκοπό την αποφυγή επιπλέον υπολογιστικού κόστους, τα δεδομένα ανάμεσα στο Engine και το υψηλότερο επίπεδο στην ιεραρχία της αρχιτεκτονικής, το World, συγχρονίζονται περιοδικά. Επιπλέον η επεξεργασία και η αποθήκευση των Events που έχουν δημιουργηθεί κατά την διάρκεια του παιχνιδιού γίνεται τοπικά σε κάποια βάση δεδομένων.

Επίσης το Engine είναι ένας μηχανισμός ελέγχου που προσφέρει διαφορετικές υπηρεσίες για κάθε παιχνίδι και δίνει την δυνατότητα ανάπτυξης διαφορετικών σεναρίων σε κάθε παιχνίδι. Η επικοινωνία ανάμεσα στα Stations και το Engine επιτυγχάνεται είτε μέσω ασυρμάτου δικτύου είτε μέσω Ethernet καλωδίου με την χρήση του Internet Protocol (IP).

3.2.4 World

Το υψηλότερο επίπεδο στην ιεραρχία της αρχιτεκτονικής, είναι το World που επι- τρέπει την διαχείριση πολλαπλών παιχνιδιών. Το συγκεκριμένο επίπεδο περιλαμβάνει το World Portal, το οποίο είναι το κεντρικό σημείο διαχείρισης του συστήματος, προσφέροντας αλληλεπίδραση με όλα τα παιχνίδια του πραγματικού κόσμου. Περιλαμβάνει επίσης την κεντρική βάση δεδομένων στην οποία αποθηκεύονται όλα τα δεδομένα που σχετίζονται με τα παιχνίδια όπως στατιστικά που αφορούν τους παίκτες και το ιστορικό παλιότερων παιχνιδιών.

Η επικοινωνία του World με το επίπεδο του Engine γίνεται μέσω της κεντρικής βάσης δεδομένων που υπάρχει στο World. Το Engine γράφει όλα τα δεδομένα από τα παιχνίδια που εξελίσσονται στην βάση δεδομένων και στην συνέχεια τα διαβάζει το World Portal. Αντιστρόφως από World Portal δημιουργούνται καινούρια παιχνίδια και νέοι παίκτες και τα δεδομένα εισάγονται στη βάση δεδομένων και στη συνέχεια μπορούν να

χρησιμοποιηθούν από το Engine ανάλογα με το παιχνίδι που εκτελείται. Στη συνέχεια αναλύεται κάθε οντότητα του σχεσιακού διαγράμματος ξεχωριστά:

- **BattleEngine:** Κάθε στιγμιότυπο ενός παιχνιδιού έχει το δικό του BattleEngine. Το BattleEngine περιέχει πληροφορίες σχετικές με το παιχνίδι, όπως τοποθεσία, μέγιστο αριθμό παικτών, την κατάσταση της εξέλιξης του παιχνιδιού, την ώρα έναρξης και λήξης του παιχνιδιού καθώς επίσης και την IP διεύθυνση του μηχανήματος στο οποίο εκτελείται. Όλες αυτές οι πληροφορίες αρχικοποιούνται πριν την έναρξη του παιχνιδιού από το World Portal.

- **Station:** Σε κάθε BattleEngine έχουν οριστεί τα Stations τα οποία θα είναι ενεργά κατά την διάρκεια του παιχνιδιού. Η οντότητα Station περιέχει το όνομα του Station, την IP διεύθυνση του μηχανήματος στο οποίο εκτελείται έτσι ώστε να μπορεί να επικοινωνήσει μαζί του το Engine, την τοποθεσία, τις ακριβείς συντεταγμένες κ.α. Κάθε Station δεν μπορεί να είναι ενεργό ταυτόχρονα σε παραπάνω από ένα Engine.

- **Avatar:** Με την εγγραφή ενός νέου παίκτη στο σύστημα δημιουργείται μια νέα εγγραφή στη βάση δεδομένων στον πίνακα Avatar. Η οντότητα Avatar περιέχει πληροφορίες σχετικές με τον παίκτη, είναι αυτή που του επιτρέπει να συμμετάσχει στα παιχνίδια και τον κάνει να ξεχωρίζει από τους άλλους παίκτες. Στον Avatar υπάρχουν και στατιστικά όσο αναφορά τα παιχνίδια που έχει χάσει ή κερδίσει.

- **Team:** Η οντότητα Team δίνει την δυνατότητα ομαδικών παιχνιδιών. Στο Team δηλώνονται ο μέγιστος αριθμός παικτών της ομάδας, το όνομα της καθώς και οι Avatars που ανήκουν στην ομάδα. Πρέπει να αναφερθεί πως οι ομάδες αφορούν στιγμιότυπα παιχνιδιών και δεν είναι ίδιες για όλα τα παιχνίδια.

- **Event:** Κάθε κίνηση που σχετίζεται με το παιχνίδι αναπαρίσταται από ένα Event. Περιέχει την περιγραφή του Event, την χρονική στιγμή που έγινε και τον τύπο του. Event. Κάθε Event που δημιουργείται αντιστοιχεί σε ένα συγκεκριμένο Avatar. Υπάρχουν δύο τύποι Event, τα MotionEvent και τα ActionEvent. Τα MotionEvent αφορούν τις κινήσεις των παικτών στο χώρο ενώ τα ActionEvent τις ενέργειες του στο παιχνίδι.

- **Item:** Τα Items είναι τα αντικείμενα που μπορεί να βρει ένας παίκτης κατά την διάρκεια του παιχνιδιού και κάθε ένα από αυτά δίνει διαφορετικές ικανότητες στον παίκτη.

- **Guardian:** Η οντότητα Guardian αφορά την συσκευή που χρησιμοποιεί ο παίκτης κατά την διάρκεια ενός παιχνιδιού. Κάθε Avatar μπορεί να χρησιμοποιεί μόνο ένα Guardian σε κάθε παιχνίδι. Το ποιος Guardian αντιστοιχεί σε κάθε Avatar γίνεται μέσω του World Portal πριν την έναρξη του παιχνιδιού. Ο Guardian περιέχει την διεύθυνση της συσκευής (MAC address), πληροφορίες για το εάν εχει αρχικοποιηθεί η συσκευή (initPhase) και το Station στο οποίο μπορεί να βρίσκεται συνδεδεμένος ο Guardian.

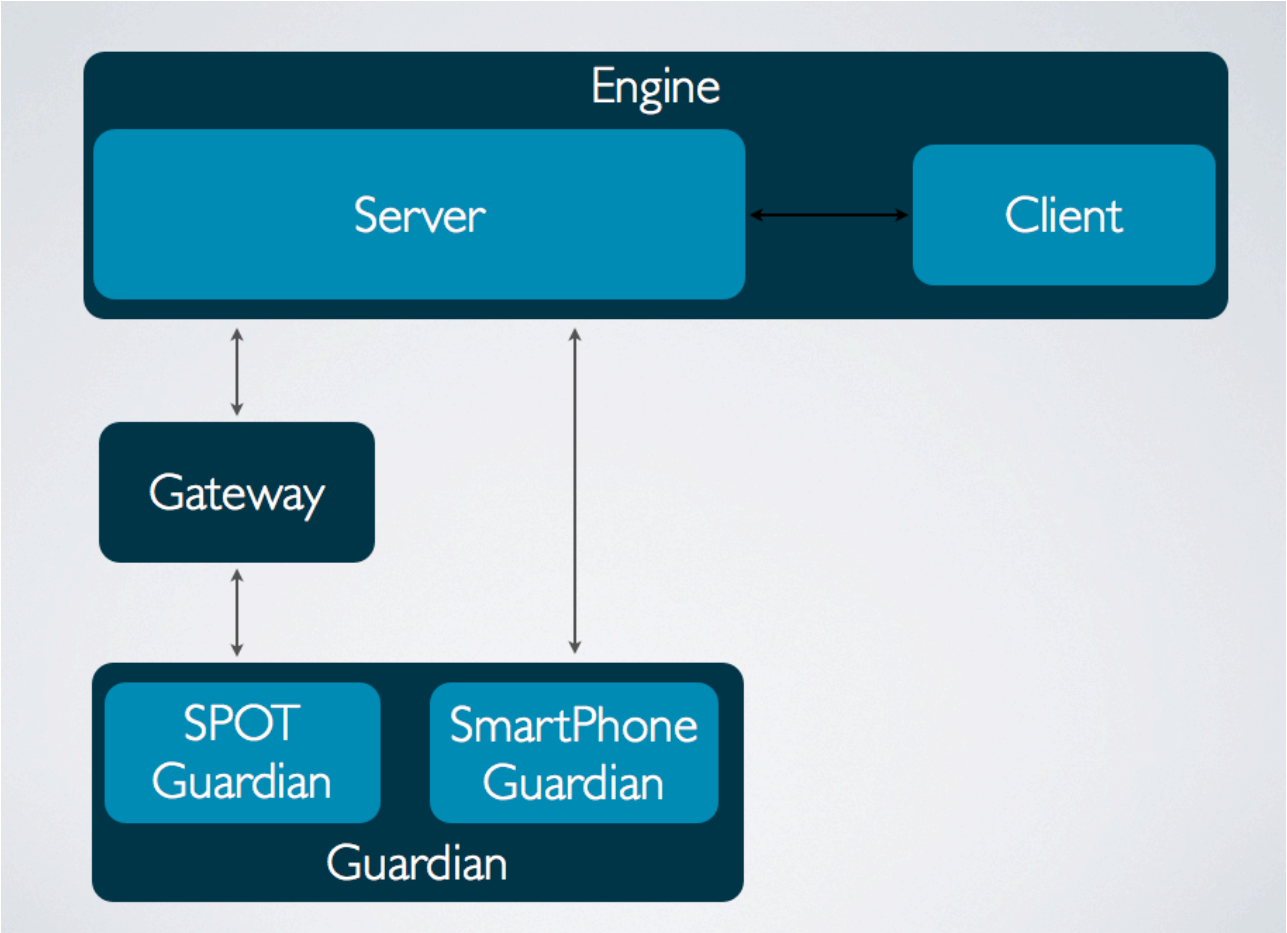
4 Αρχιτεκτονική Συστήματος

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο παρουσιάζεται μια νέα προτεινόμενη αρχιτεκτονική για την πλατφόρμα FunInNumbers η οποία βασίζεται περισσότερο σε τεχνολογίες HTML5. Οι βασικές αρχές της πλατφόρμας παραμένουν ίδιες. Σκοπός είναι μια πλατφόρμα που χαρακτηρίζεται από επεκτασιμότητα, και υποστήριξη πολλαπλών συσκευών.

Με την ανάπτυξη της τεχνολογίας όμως, η χρήση προσωπικών συσκευών με αυξημένη υπολογιστική ισχύ, και δυνατότητες συνδεσιμότητας (smartphones, tablets κλπ.) ολοένα και αυξάνεται. Έτσι, τον παρόν σύστημα εκμεταλλεύεται αυτή τη μεγάλη χρήση κινητών τηλεφώνων, και παρέχει υποστήριξη για αυτά, επιπρόσθετα με τα ασύρματα δίκτυα αισθητήρων. Επίσης σήμερα οι δυνατότητες συνδεσιμότητας στον Παγκόσμιο Ιστό είναι τεράστιες, με πολύ υψηλές ταχύτητες και τεράστια κάλυψη. Το νέο σύστημα εκμεταλλεύεται αυτές τις δυνατότητες και στηρίζεται σε τεχνολογίες του Παγκόσμιου Ιστού με σκοπό να ενώσει πιο εύκολα παίκτες από ολόκληρο τον κόσμο, με διαφορετικές συσκευές, με σκοπό να παίξουν τα ίδια διαδραστικά παιχνίδια, ταυτόχρονα.

Η νέα αρχιτεκτονική του συστήματος αποτελείται από 3 επίπεδα, που λειτουργούν ανεξάρτητα και επικοινωνούν μεταξύ τους, όπου το καθένα μπορεί να χωρίζεται και αυτό σε υποσυστήματα, και απεικονίζεται στο παρακάτω σχήμα:



Εικόνα 4.1 Νέα Αρχιτεκτονική του Συστήματος

4.2 Guardian

Το επίπεδο Guardian αποτελεί το χαμηλότερο επίπεδο της προτεινόμενης αρχιτεκτονικής, και αποτελείται από τις συσκευές που χρησιμοποιούν οι χρήστες ως χειριστήρια κατά τη διάρκεια των παιχνιδιών. Στην ουσία το επίπεδο Guardian αποτελείται από το λογισμικό που εκτελείται σε αυτές τις συσκευές και αποτελεί τη διεπαφή του συστήματος με το χρήστη. Ο χρήστης αλληλεπιδρά με αυτές τις συσκευές, χρησιμοποιώντας τους αισθητήρες που αυτές διαθέτουν ανάλογα με τους κανόνες του κάθε παιχνιδιού. Για την ακρίβεια η πλατφόρμα που προτείνεται βασίζεται αρκετά στις κινήσεις που κάνει ο κάθε παίκτης με την συσκευή, οπότε υπάρχει ένα υποσύστημα το οποίο συλλέγει και επεξεργάζεται τα δεδομένα που προκύπτουν από τον αισθητήρα επιτάχυνσης της συσκευής και αναγνωρίζει συγκεκριμένες

κινήσεις (Gestures) οι οποίες αντιστοιχούν σε ενέργειες που σχετίζονται με το παιχνίδι. Έτσι δημιουργούνται τα λεγόμενα Events, τα οποία προωθούνται στα ανώτερα επίπεδα. Το επίπεδο Guardian χωρίζεται σε 2 υποεπίπεδα, το SPOT Guardian, και το Smartphone Guardian.

4.1.1 SPOT Guardian

Το συγκεκριμένο επίπεδο της αρχιτεκτονικής του συστήματος αφορά στην υλοποίηση του επιπέδου Guardian στις συσκευές SunSPOT που περιγράφονται παραπάνω. Κάθε συσκευή εκτελεί ένα λογισμικό που αναγνωρίζει Gestures, τα αντιστοιχεί σε Events που σχετίζονται με το τρέχον παιχνίδι, και τα προωθεί στο επίπεδο Gateway. Η επικοινωνία ανάμεσα στα δύο αυτά επίπεδα γίνεται με Datagram Packets με χρήση του πρωτοκόλλου IEEE 802.15.4.

4.1.2 SmartPhone Guardian

Το επίπεδο Smartphone Guardian σχετίζεται με την υλοποίηση του επιπέδου Guardian σε κινητά τηλέφωνα με λειτουργικό σύστημα Android ή iOS. Κάθε χρήστης χρησιμοποιεί το κινητό του τηλέφωνο για να αλληλεπιδράσει με το παιχνίδι, κάνοντας κινήσεις με αυτό, ενώ η συσκευή λειτουργεί με παρόμοιο τρόπο με τα SunSPOTs χρησιμοποιώντας όμως και την οθόνη, τα ηχεία, και το σύστημα δόνησης του κινητού τηλεφώνου, για να ειδοποιήσει το χρήστη για συμβάντα που σχετίζονται με το παιχνίδι (πχ. για κάθε σωστή κίνηση του χρήστη ακούγεται κάποιος ήχος). Το επίπεδο αυτό επικοινωνεί απευθείας με το επίπεδο Engine Server μέσω WebSockets.

4.3 Gateway

Το επίπεδο Gateway αποτελεί το επίπεδο που ενώνει τους SPOT Guardians με το επίπεδο Engine. Πρακτικά πρόκειται για το λογισμικό που εκτελείται σε ένα μηχάνημα συνδεδεμένο με ένα Sun SPOT base station και δέχεται τα μηνύματα των SPOT Guardians. Αφού μετατρέψει τα Events στην κατάλληλη μορφή, τα προωθεί στο επίπεδο Engine. Πολλά Gateways μπορούν να επικοινωνούν με ένα Engine, έτσι παίκτες που είναι σε διαφορετικά σημεία του κόσμου, μπορούν να παίξουν το ιδιο παιχνίδι χρησιμοποιώντας SunSPOTs, επικοινωνώντας με το ίδιο Engine.

4.4 Engine

Το Engine αποτελεί το υψηλότερο επίπεδο της αρχιτεκτονικής του συστήματος. Χωρίζεται σε δύο υποσυστήματα, το Engine Server, και το Engine Client, και είναι υπεύθυνο για την εφαρμογή των κανόνων του κάθε παιχνιδιού, και την απεικόνιση του παιχνιδιού στους παίκτες.

4.4.1 Engine Server

Το επίπεδο Engine Server είναι υπεύθυνο για την υλοποίηση και εφαρμογή των κανόνων του κάθε παιχνιδιού. Σε αυτό το επίπεδο αποθηκεύονται όλες οι πληροφορίες που αφορούν στην ομαλή διεξαγωγή του κάθε παιχνιδιού, και συγκεντρώνονται τα Events που προκαλούνται από τους παίκτες. Κάθε event που επεξεργάζεται ο Server έχει σαν αποτέλεσμα να αλλάξει η τρέχουσα κατάσταση του παιχνιδιού (Game State). Επίσης ο Server αποθηκεύει όλους όσους έχουν συνδεθεί σε αυτόν. Έτσι γνωρίζει ανα πάσσα στιγμή πόσοι παίκτες παίζουν το παιχνίδι, και πόσοι Engine Clients έχουν συνδεθεί. Έτσι μόλις αλλάξει το Game State, αφού αποθηκευτεί πρέπει να σταλεί σε όλους τους Engine Clients.

4.4.2 Engine Client

Το Engine Client αφορά στην απεικόνιση του Game State στους παίκτες. Εκτελείται σε Περιηγητές Ιστού, επικοινωνεί με τον Engine Server, και μόλις λάβει μήνυμα με καινούριο game state το απεικονίζει.

4.5 Σύνοψη

Ένα παιχνίδι συντονίζεται και ξεκινά από το επίπεδο Engine. Ο Engine server ενός παιχνιδιού εκτελείται συνεχόμενα σε έναν WebServer. Για να παίξει ένας παίκτης, θα πρέπει αρχικά να ανοίξει έναν Περιηγητή Ιστού και να ανοίξει την ιστοσελίδα που αποτελεί τον Engine Server. Ύστερα πρέπει να ανοίξει την εφαρμογή του παιχνιδιού στο κινητό του τηλέφωνο, ή να τρέξει ένα Gateway στον υπολογιστή του, και να συνδεθεί χρησιμοποιώντας ένα SunSPOT.

Σημειώνεται ότι δίνεται η δυνατότητα στους παίκτες, να εκτελέσουν τοπικά ένα Engine Server, σε κάποιον WebServer στο τοπικό δίκτυο. Έτσι χρησιμοποιώντας το μπορούν να συνδεθούν στον Server.

5 Ζητήματα Υλοποίησης

5.1 Εισαγωγή

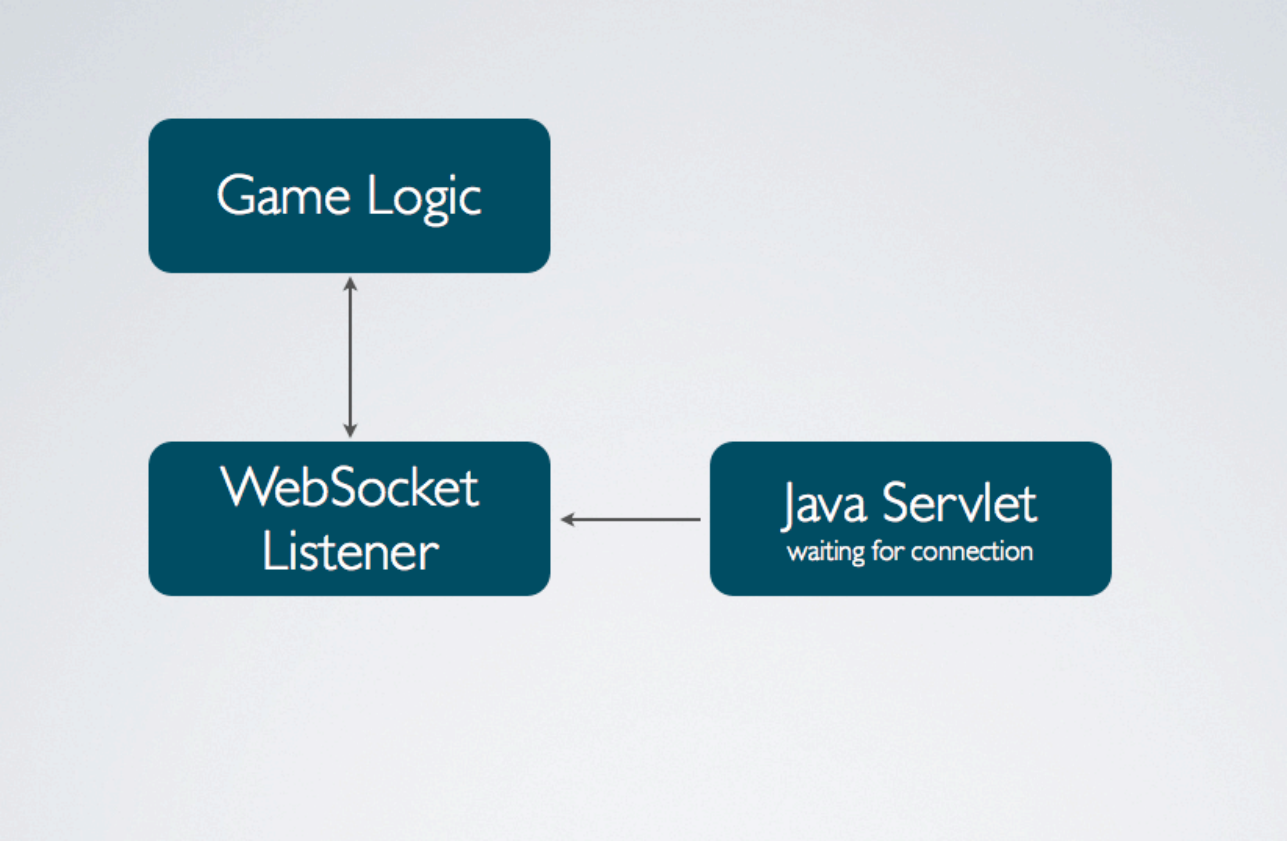
Σε αυτό το κεφάλαιο παρουσιάζονται κάποια ζητήματα υλοποίησης που προέκυψαν κατά την ανάπτυξη του συστήματος, καθώς και οι σχεδιαστικές αποφάσεις που λήφθηκαν για την αντιμετώπιση τους. Κυρίως αφορούν σε ζητήματα τεχνολογιών που χρησιμοποιήθηκαν για την υλοποίηση του κάθε επιπέδου, ζητήματα επικοινωνίας ανάμεσα στα επίπεδα, πρωτόκολλα επικοινωνίας για την ομαλή λειτουργία του συστήματος κλπ.

5.2 Engine

Το επίπεδο Engine αποτελείται από 2 υποεπίπεδα, έναν server και έναν client. Η υλοποίηση του επιπέδου Engine στηρίζεται σε μια εφαρμογή ιστού (Web App) βασισμένη στο Google Web Toolkit που αναλύεται παραπάνω. Στην ουσία ο Engine Server εκτελείται στον ο εξυπηρετητή ιστού που εκτελείται ο server του GWT Web App, και ο Engine Client αποτελεί τον client του Web App και εκετελείται στους περιηγητές ιστού των παικτών. Στα πλαίσια της εργασίας χρησιμοποιήθηκε το GWT v2.5.0 για την ανάπτυξη του web app, και ο server εκτλείται σε έναν Resin v4.0.23 web server **[12]**.

5.2.1 Δομή Engine Server

Η εσωτερική δομή του επιπέδου Engine Server φαίνεται στο παρακάτω σχήμα.



Εικόνα 5.1 Εσωτερική Δομή του υποσυστήματος Engine Server

Παραπάνω φαίνονται τα 3 υποσυστήματα που περιέχει το Engine server επίπεδο. Οι λειτουργίες που εξυπηρετεί το κάθε υποσύστημα είναι οι εξής:

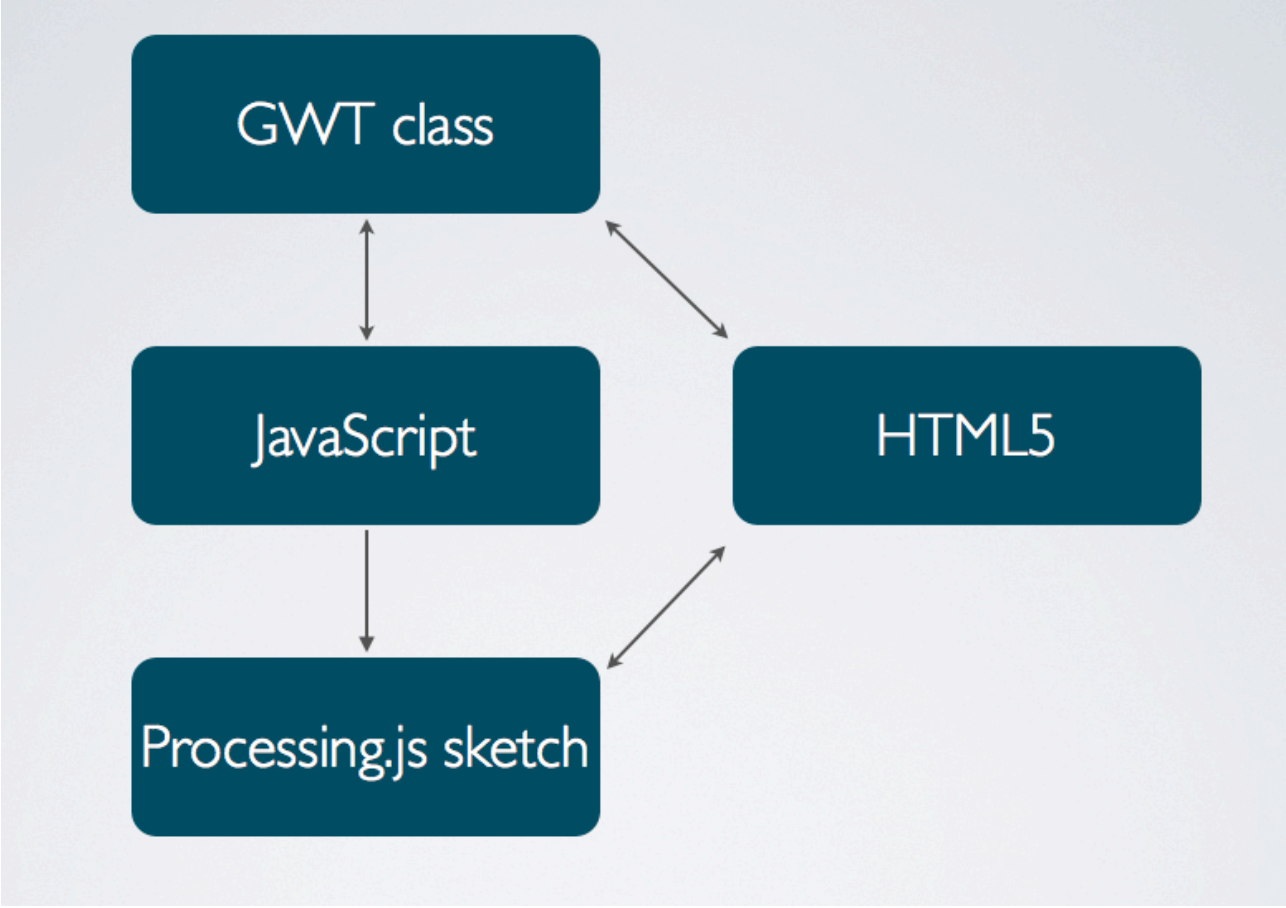
- **Java Servlet**: Πρόκειται για ένα service το οποίο εκτελείται συνεχώς και περιμένει εισερχόμενα http αιτήματα. Όταν δεχθεί ένα αίτημα, εξετάζει εάν είναι websocket request, με το σωστό subprotocol, και αν είναι το προωθεί στο υποσύστημα WebSocket Listener.

- **WebSocket Listener**: Το υποσύστημα WebSocket Listener είναι υπεύθυνο για την επεξεργασία τον μηνυμάτων που φθάνουν στο επίπεδο Engine Server μέσω WebSockets, που έχουν ως αποτέλεσμα την αλλαγή της τρέχουσας κατάστασης του παιχνιδιού, η οποία αποθηκεύεται στο υποσύστημα Game Logic, καθώς και την αποστολή των απαραίτητων μηνυμάτων μέσω WebSockets στα υπόλοιπα επίπεδα της αρχιτεκτονικής που επικοινωνούν με το Engine Server.

- **Game Logic**:Το υποσύστημα Game Logic αποθηκεύει όλες τις πληροφορίες που απαιτούνται για την διατήρηση της τρέχουσας κατάστασης του κάθε παιχνιδιού. Το επίπεδο WebSocket Listener προσπελαύνει αυτή την πληροφορία και την μεταβάλλει ανάλογα με τα μηνύματα που δέχεται από τους παίκτες. Επίσης, το Game Logic είναι ένα Observable αντικείμενο, ενώ το WebSocket Listener είναι Observer. Έτσι, οποτεδήποτε χρειάζεται να σταλεί κάποια νέα πληροφορία σε κάποιο άλλο επίπεδο, το Game Logic ειδοποιεί το WebSocket Listener να στείλει αυτή την πληροφορία.

5.2.2 Δομή Engine Client

Το επίπεδο Engine Client είναι υπεύθυνο για την απεικόνιση της κατάστασης του παιχνιδιού στους παίκτες. Πρόκειται για τον client της εφαρμογής ιστού GWT και εκτελείται φυσικά σε περιηγητές ιστού. Στην εικόνα που ακολουθεί φαίνεται η δομή του επιπέδου.



Εικόνα 5.2 Εσωτερική Δομή του υποσυστήματος Engine Server

Οι λειτουργίες που παρέχει το κάθε υποσύστημα είναι οι εξής:

- **HTML5**: Με τη χρήση της HTML5, περιγράφεται η οπτική απεικόνιση του παιχνιδιού, και ορίζονται τα διάφορα στοιχεία που την συνιστούν.

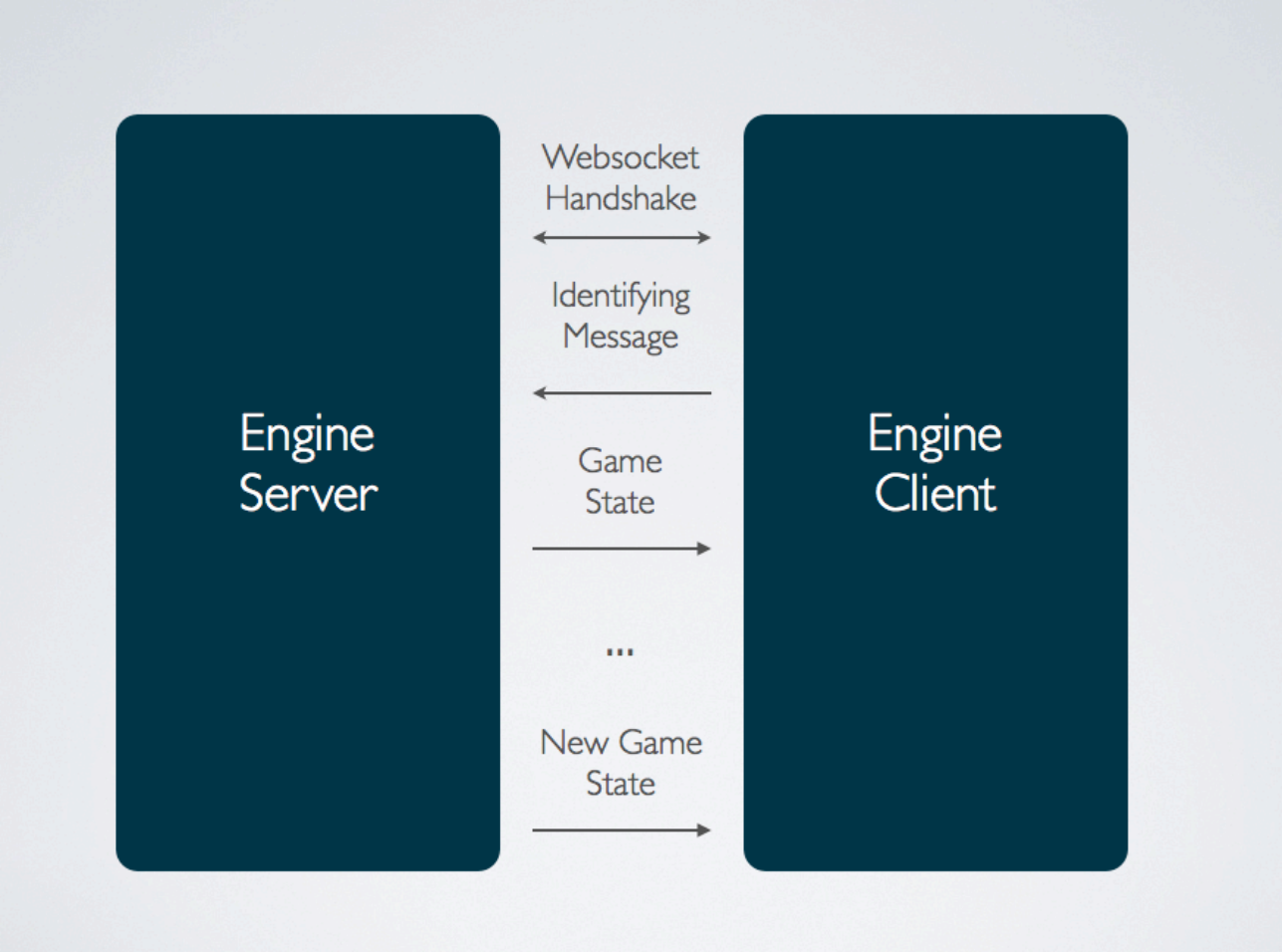
- **Processing.js sketch**: Αυτό το υποσύστημα είναι υπεύθυνο για την παραγωγή και απεικόνιση των γραφικών του παιχνιδιού. Ζωγραφίζει τα κινούμενα σχέδια που εξυπηρετούν το παιχνίδι, πάνω σε ένα στοιχείο canvas που έχει οριστεί στο αρχείο της HTML5, που περιγράφηκε παραπάνω.

- **Javascript**: Το υποσύστημα αυτό είναι υπεύθυνο για την αρχικοποίηση των websockets, που είναι απαραίτητα για την επικοινωνία με τον Engine Server. Επίσης αποτελεί ένα δίαυλο επικοινωνίας ανάμεσα στο GWT Class και το Processing.js sketch.

- **GWT Class**: Το κυρίως μέρος του επιπέδου Engine Client. Αποτελεί τον συντονιστή της όλης εφαρμογής. Είναι υπεύθυνο για την επεξεργασία των μηνυμάτων που δέχεται το επίπεδο από τον server, και την απεικόνιση του αποτελέσματος τους. Χρησιμοποιεί το JSNI για να επικοινωνήσει με την Javascript και κατά συνέπεια με την Processing.js

5.2.2. Πρωτόκολλο επικοινωνίας μεταξύ Engine Client - Engine Server

Για να εγκατασταθεί σωστά μια σύνδεση ανάμεσα στα 2 αυτά υποσυστήματα του επιπέδου Engine πρέπει να διατηρηθεί ένα συγκεκριμένο πρωτόκολλο. Το πρωτόκολλο αυτό φαίνεται στο παρακάτω σχήμα.



Εικόνα 5.3 Πρωτόκολλο επικοινωνίας ανάμεσα στα υποσυστήματα Engine Server & Engine Client

Στην ουσία ο Engine Server, είναι και εξυπηρετητής του WebSocket. Έτσι, μόλις ένα Engine Client επιχειρήσει να συνδεθεί εκτελείται σύμφωνα με το πρωτόκολλο WebSocket το handshake που χρειάζεται για να εγκατασταθεί η σύνδεση. Στη συνέχεια ο Engine Client, πρέπει να στείλει ένα μήνυμα για να ειδοποιήσει τον Engine Server οτι είναι client. Ο Engine Server διατηρεί μια λίστα με όλους τους συνδεδεμένους πελάτες, και έτσι γράφει τον Client σε αυτή τη λίστα, μαζί με το αναγνωριστικό του, που προκύπτει από την σύνδεση που έχει εγκατασταθεί μεταξύ τους. Ύστερα ο Server στέλνει στον Client την παρούσα κατάσταση του παιχνιδιού, ώστε αυτή να μπορεί να απεικονιστεί κατάλληλα στον Περιηγητή Ιστού των παικτών.

Μόλις συμβεί ένα γεγονός, που θα αλλάξει την παρούσα κατάσταση του παιχνιδιού, ο Server οφείλει να ενημερώσει τον Client ανατρέχοντας στη

λίστα που αναφέρθηκε παραπάνω, και στέλνοντας ένα μήνυμα με την περιγραφή της νέας.

5.3 Gateway

Το επίπεδο Gateway αφορά στο λογισμικό που εκτελείται σε ένα υπολογιστικό μηχάνημα συνδεδεμένο με το SunSPOT Base Station, και λειτουργεί σαν 'γέφυρα' ανάμεσα στους παίκτες που παίζουν χρησιμοποιώντας SunSPOT και στο Game Engine. Η υλοποίηση του είναι σχετικά απλή καθώς έχει ως σκοπό να δέχεται τα Events των SPOT Guardians να τα μετασχηματίζει την κατάλληλη μορφή, και να τα προωθεί στον Engine Server. Περισσότερες πληροφορίες για αυτή τη διαδικασία, τεχνολογίες και πρωτόκολλα που χρησιμοποιούνται αναφέρονται παρακάτω, στην ενότητα 5.5.

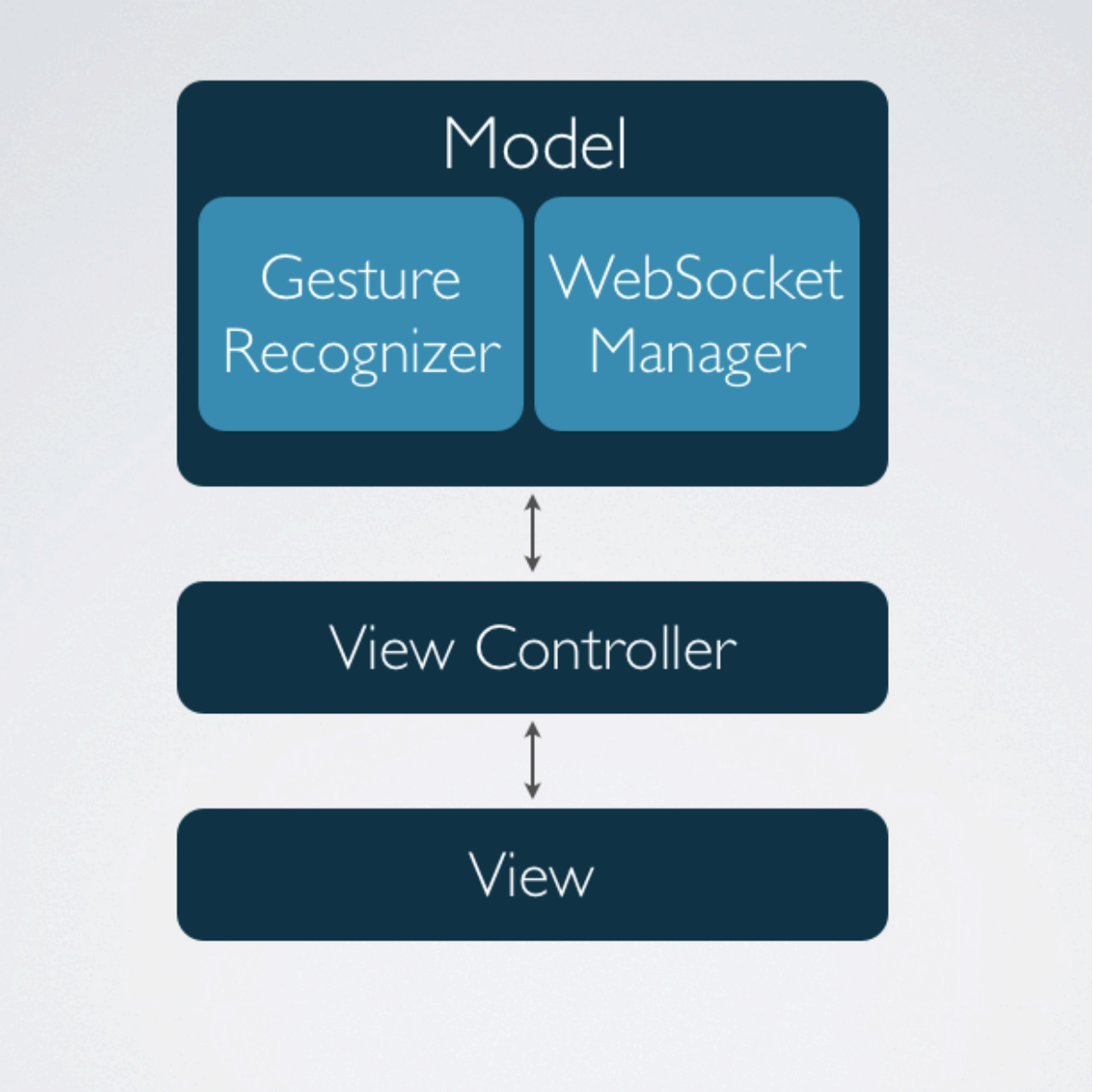
5.4 Guardian

5.4.2 SmartPhone Guardian

Στα πλαίσια της εργασίας επιλέχθηκε να υλοποιηθεί το επίπεδο αυτό στα δύο πιο δημοφιλή λειτουργικά συστήματα κινητών συσκευών: iOS και Android.

5.4.2.1 iOS

Η αρχιτεκτονική της εφαρμογής του Guardian για συσκευές που τρέχουν iOS ακολουθεί το MVC (Model-View-Controller) μοντέλο προγραμματισμού. Τα υποσυστήματα που εκτελούνται, και το πεδίο του MVC στο οποίο ανήκουν φαίνονται στο παρακάτω σχήμα.



Εικόνα 5.4 Εσωτερική δομή iOS SmartPhone Guardian

Το View είναι το υποσύστημα της εφαρμογής που αποτελεί τη διεπαφή του χρήστη με την εφαρμογή. Είναι υπεύθυνο για την αναπαράσταση των δεδομένων της εφαρμογής στο χρήστη.

Ο View Controller είναι εκείνος που ευθύνεται για την επεξεργασία των πράξεων του χρήστη στο View, και στην ουσία λέει στο View τι να δείξει στην οθόνη της συσκευής.

Το επίπεδο Model αφορά στις υπηρεσίες που παρέχονται από την εφαρμογή, και δεν φαίνονται άμεσα από τη χρήση.

Το υποσύστημα WebSocket Manager είναι υπεύθυνο για την αρχικοποίηση της σύνδεσης WebSocket με τον Engine Server, την επεξεργασία των εισερχόμενων μηνυμάτων, καθώς και την αποστολή μηνυμάτων που προκύπτουν από πράξεις του χρήστη.

Το υποσύστημα Gesture Recognizer χρησιμοποιεί τα δεδομένα του αισθητήρα επιτάχυνσης που διαθέτει η συσκευή, και αναγνωρίζει κινήσεις που αυτός κάνει, για παράδειγμα παλλινδρομική κίνηση στον οριζόντιο άξονα, απότομη κίνηση προς τα εμπρός κλπ. Ο τρόπος με τον οποίο γίνεται αυτό είναι με τη χρήση του αντικειμένου CMMotionManager που παρέχεται από το λειτουργικό. Ο CMMotionManager διαθέτει 3 τιμές που αναπαριστούν τις τιμές του αισθητήρα επιτάχυνσης της συσκευής στους 3 άξονες. Με δειγματοληψία αυτών, και σύγκριση διαδοχικών τιμών, προκύπτει η αναγνώριση κινήσεων του παίκτη.

5.4.2.2 Android

Η υλοιποίηση σε Android είναι αρκετά παρόμοια με αυτή του iOS. Υπάρχει ένα σύνολο από Activities, με το main Activity να είναι παρόμοιο με το View του iOS. Τα υπόλοιπα Activities αποτελούν το Menu της εφαρμογής.

Και εδώ υπάρχει ένα υποσύστημα Gesture Recognizer το οποίο λειτουργεί με παρόμοιο τρόπο, χρησιμοποιώντας τις τιμές του αισθητήρα επιτάχυνσης για τους 3 άξονες. Επίσης υπάρχει ένα υποσύστημα WebSocket Manager, που διαχειρίζεται το WebSocket, και είναι υπεύθυνο για το Serialization των Events και την αποστολή τους στον Server.

5.4.3 SPOT Guardian

Η υλοποίηση των SPOT Guardians είναι ακριβώς η ίδια με την υλοποίηση των Guardians στην προηγούμενη αρχιτεκτονική του FunInNumbers. Ο τρόπος με τον οποίο επικοινωνεί ένας Guardian με το υπόλοιπο σύστημα είναι το υλοποιημένο Echo Protocol, και διαθέτει επίσης το Delay Tolerance Service.

5.5 Θέματα επικοινωνίας ανάμεσα στα επίπεδα της αρχιτεκτονικής

Μια πρόκληση στην σχεδίαση της αρχιτεκτονικής ήταν η επικοινωνία ανάμεσα στα διάφορα επίπεδα της αρχιτεκτονικής. Ο διαχωρισμός του συστήματος στα διάφορα υποσυστήματα έγινε με σκοπό την κλιμακωσιμότητα του συστήματος, και την ευκολία υποστήριξης νέων τύποι συσκευών.

5.5.1 Κανάλι επικοινωνίας του Engine Server με τα υπόλοιπα επίπεδα

Παρατηρούμε από την αρχιτεκτονική πως πολύ βασικό ρόλο στο σύστημα έχει ο Engine Server. Για να λειτουργήσει το σύστημα σαν σύνολο χρειάζεται να επικοινωνήσει με τρία ακόμη υποσυστήματα: τον Engine Client, το Gateway, και τα Smartphone Guardians. Έτσι επιλέχθηκε ένα κοινό κανάλι επικοινωνίας που ενώνει όλα αυτά τα υποσυστήματα με τον Server: τα WebSockets.

Βασικό πλεονέκτημα αυτής της επιλογής είναι πως όλα τα δομικά στοιχεία της προτεινόμενης αρχιτεκτονικής υποστηρίζουν την τεχνολογία των WebSockets. Επίσης η υλοποίηση τους είναι αρκετά απλή, και ξεκάθαρη.

Πρακτικά αυτό που γίνεται είναι ο Engine Server να γίνεται Server του WebSocket Service, και όλα τα υπόλοιπα υποσυστήματα να γίνονται clients. Για να διαχωρίζει ο Server σε ποιο επίπεδο ανήκει ο κάθε πελάτης, δημιουργήθηκαν κάποια πολύ απλά πρωτόκολλα τα οποία εξηγούνται παρακάτω.

5.5.2 Ομοιομορφία επεξεργασίας των Game Event

Ένα άλλο ζήτημα που προέκυψε αφορά στην επεξεργασία των Events που δημιουργούνται στο επίπεδο των Guardians, από τον τελικό αποδέκτη, τον Engine Server. Σκοπός της πλατφόρμας είναι η δυνατότητα παίκτες από όλο τον κόσμο να μπορούν να παίξουν μαζί σε διαδραστικά παιχνίδια, χρησιμοποιώντας δίκτυα αισθητήρων και κινητά τηλέφωνα. Άρα έχει μεγάλη σημασία να δημιουργηθεί ένα σύστημα εύκολα κλιμακώσιμο στον αριθμό των συσκευών που υποστηρίζονται. Έτσι χρειάζεται να βρεθεί ένας τρόπος, η επεξεργασία των events που δημιουργούνται από τις συσκευές των χρηστών στο επίπεδο Engine Server να γίνεται ακριβός με τον ίδιο τρόπο. Αυτός ο τρόπος είναι το Serialization των δεδομένων, με χρήση των Google Protocol Buffers. Επιλέχθηκαν τα Google Protocol Buffers υποστηρίζονται από πολλές γλώσσες προγραμματισμού, και είναι αρκετά εύκολα στη χρήση τους.

Οι Smartphone Guardians κάνουν serialize το Event πριν το στείλουν στο Engine Layer, ακριβώς με τον ίδιο τρόπο με τον οποίο το επίπεδο Gateway κάνει Serialize τα events που λαμβάνει από τα SunSPOTs. Έτσι ο Server επεξεργάζεται τα μηνύματα και αλλάζει την τρέχουσα κατάσταση του παιχνιδιού με τον ίδιο τρόπο σε κάθε περίπτωση χωρίς να γνωρίζει την συσκευή που χρησιμοποίησε ο παίκτης για να κάνει αυτή την κίνηση. Για να υποστηριχθεί μια νέα συσκευή λοιπόν, αρκεί να υλοποιηθεί με τέτοιο τρόπο ώστε να στέλνει μηνύματα της ίδιας μορφής στον Engine Server.

Η μορφή του Event που προκαλούν οι χρήστες είναι η ίδια με αυτήν που είχε σχεδιαστεί για την αρχική πλατρφόρμα. Ο κώδικας που γίνεται compile με το εργαλείο proto, ώστε να προκύψουν οι αντίστοιχες κλάσεις στις γλώσσες είναι ο εξής:

```
message ProtoEvent {
  required int32 id = 1;
  required int32 teamId = 2;
  required int32 guardianCounter=3;
  required string type = 4;
  required string description = 5;
  required int32 battleEngine = 6;
  required int64 timeStamp = 7;
  required int32 stationId = 8;
  optional string guardianIp= 9;
}
```

Εικόνα 5.5 Περιγραφή Protocol Buffer Event

Οι απαντήσεις που δίνει ο Server στο Gateway και στους Smartphone Guardians γίνεται με απλά μηνύματα Text. Επίσης όλα τα άλλα μηνύματα που στέλνονται ανάμεσα στον Server και στα 3 layers που αυτός επικοινωνεί, και διαμορφώνουν τα πρωτόκολλα που φαίνονται παρακάτω είναι απλά μηνύματα Text. Αυτό έγινε διότι όλα αυτά τα μηνύματα από τη φύση τους δεν είχαν πολύπλοκη δομή σαν τα Events, και επειδή κείμενο μπορεί πολύ εύκολα να σταλεί μέσω των WebSockets. Επίσης αυτό αποτελεί μια εύκολη μέθοδο διαχωρισμού των Events από τα άλλα μηνύματα από τον Server, αφού για

κάθε Event εκτελείται η ειδική συνάρτηση onBinaryMessage(), ενώ σε κάθε άλλη περίπτωση εκτελείται η onTextMessage().

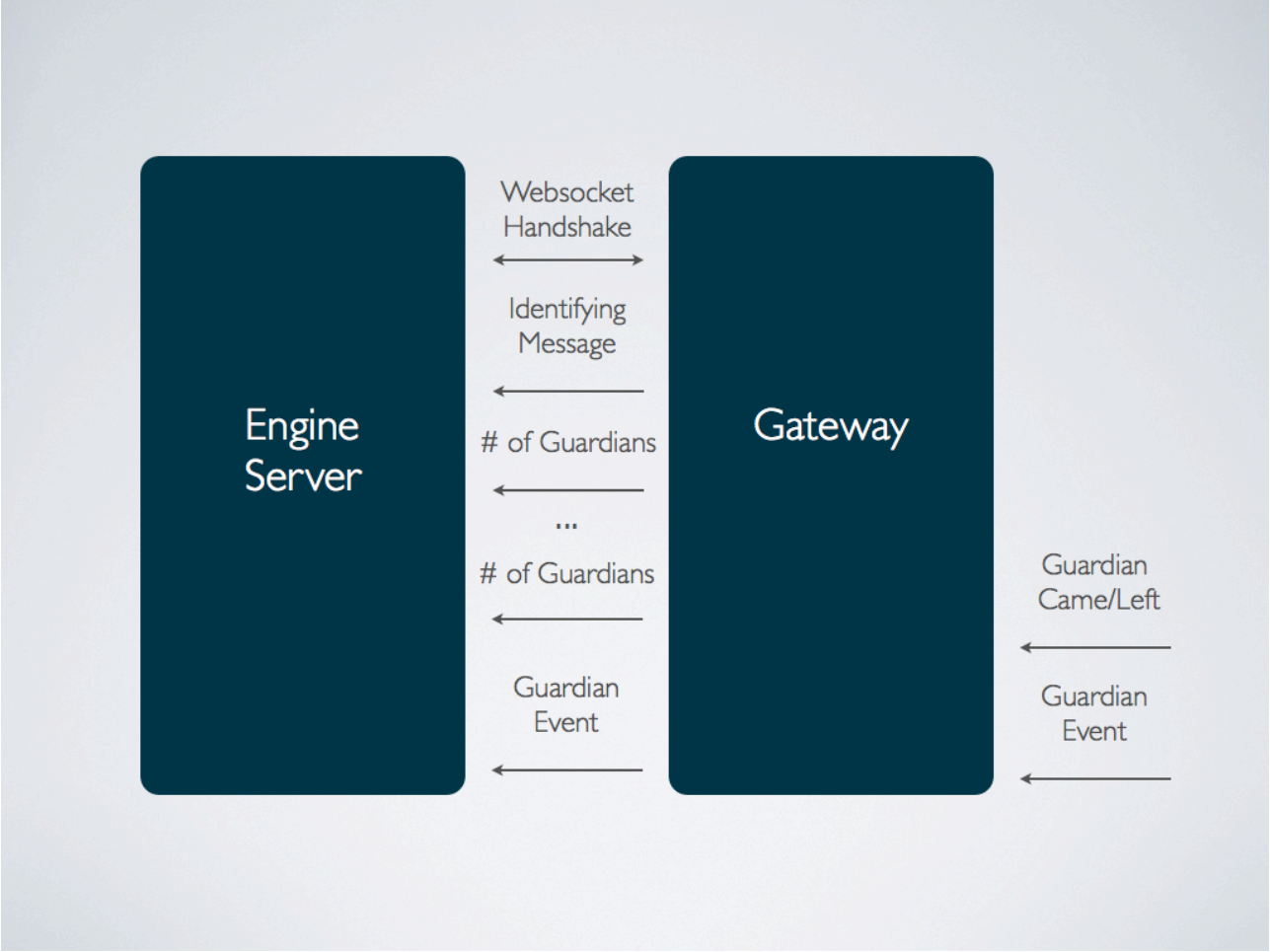
Η επικοινωνία ανάμεσα στα SpotStation και στο Gateway γίνεται μέσω του IEEE 802.15.4, και τα Events από τα SunSPOTS στο Gateway προωθούνται με τη μορφή Datagram Packets. Το πρωτόκολλο επικοινωνίας που χρησιμοποιείται είναι το Echo Protocol, που είχε υλοποιηθεί για την επικοινωνία των επιπέδων Guardian και Station της προηγούμενης αρχιτεκτονικής.

5.5.3 Πρωτόκολλα Επικοινωνίας

Όπως αναφέρεται παραπάνω, ο Server πρέπει να διαχωρίσει τους πελάτες που επικοινωνούν μαζί του μέσω του κοινού καναλιού. Έτσι εκτός από το πρωτόκολλο μεταξύ Engine Server και Engine Client που εξηγείται παραπάνω, υπάρχουν 2 ακόμη παρόμοια πρωτόκολλα που αφορούν στην επικοινωνία του Server με τα 2 άλλα επίπεδα.

5.5.3.1 Πρωτόκολλο Engine Server - Gateway

Το πρωτόκολλο περιγράφεται συνοπτικά από την παρακάτω εικόνα.



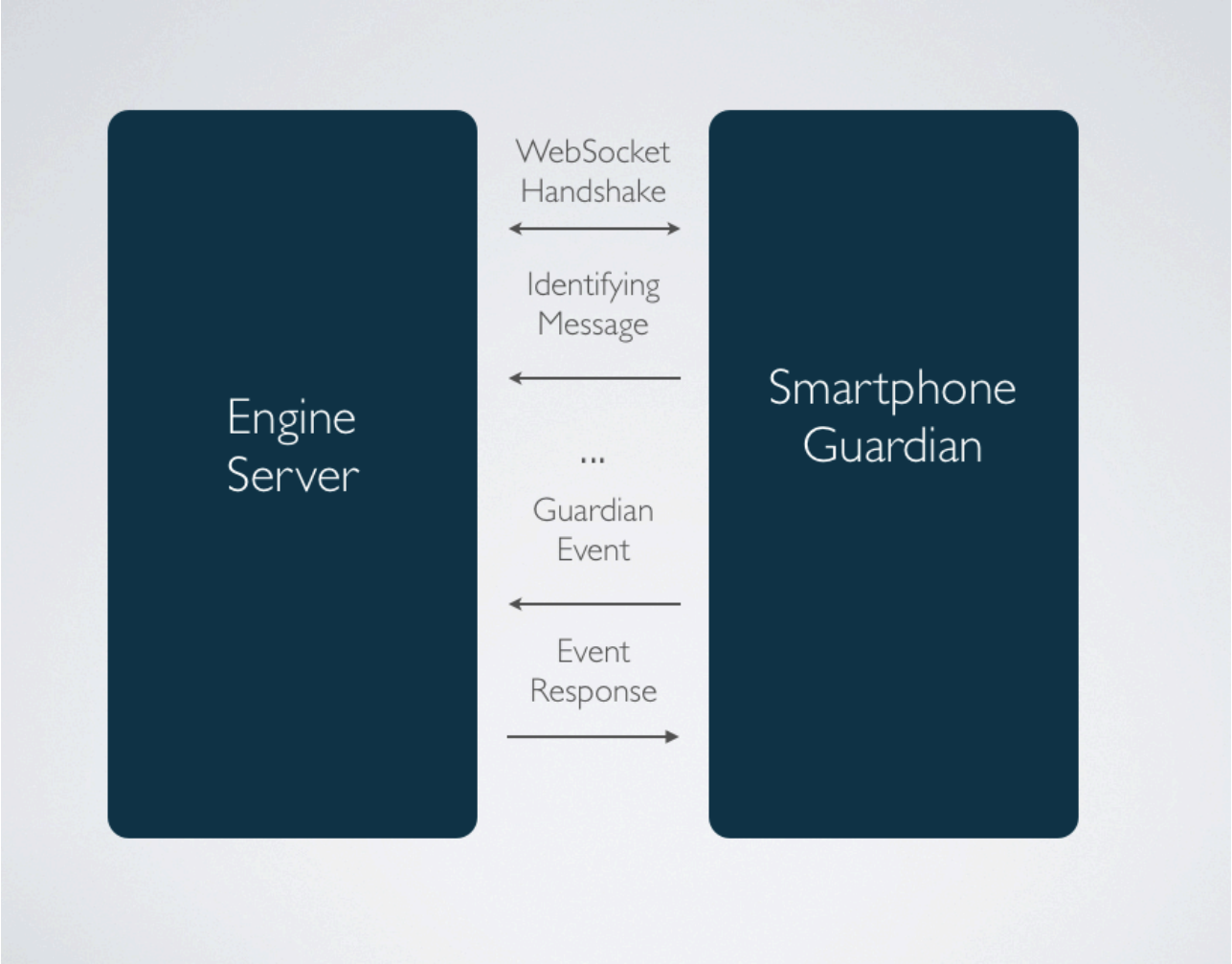
Εικόνα 5.6 Πρωτόκολλο Engine Server - Gateway

Αφού γίνει το HandShake του WebSocket και εγκατασταθεί η σύνδεση, το Gateway στέλνει ένα μήνυμα για να ενημερώσει για την ταυτότητα του. Τότε ο Server αποθηκεύει το Gateway και το αναγνωριστικό του σε μια λίστα που διαθέτει με όλα τα Gateway που έχουν συνδεθεί. Μετά το Gateway πρέπει να στείλει ένα μήνυμα με τον αριθμό των Guardians που είναι συνδεδεμένα σε αυτό. Κάθε φορά που ένας νέος Guardian συνδέεται με το Gateway, ή κάποιος άλλος χάνει το connection, τότε το Gateway πρέπει να ενημερώσει τον Server για το νέο αριθμό των συνδεδεμένων Guardians.

Αν κάποιος Guardian δημιουργήσει κάποιο event, τότε αυτό προωθείται στο Gateway, το Gateway κάνει Serialize με χρήση Google Protocol Buffers τα δεδομένα και τα στέλνει στον Server.

5.5.3.1 Πρωτόκολλο Engine Server - Smartphone Guardian

Το πρωτόκολλο περιγράφεται συνοπτικά από το παρακάτω σχήμα.



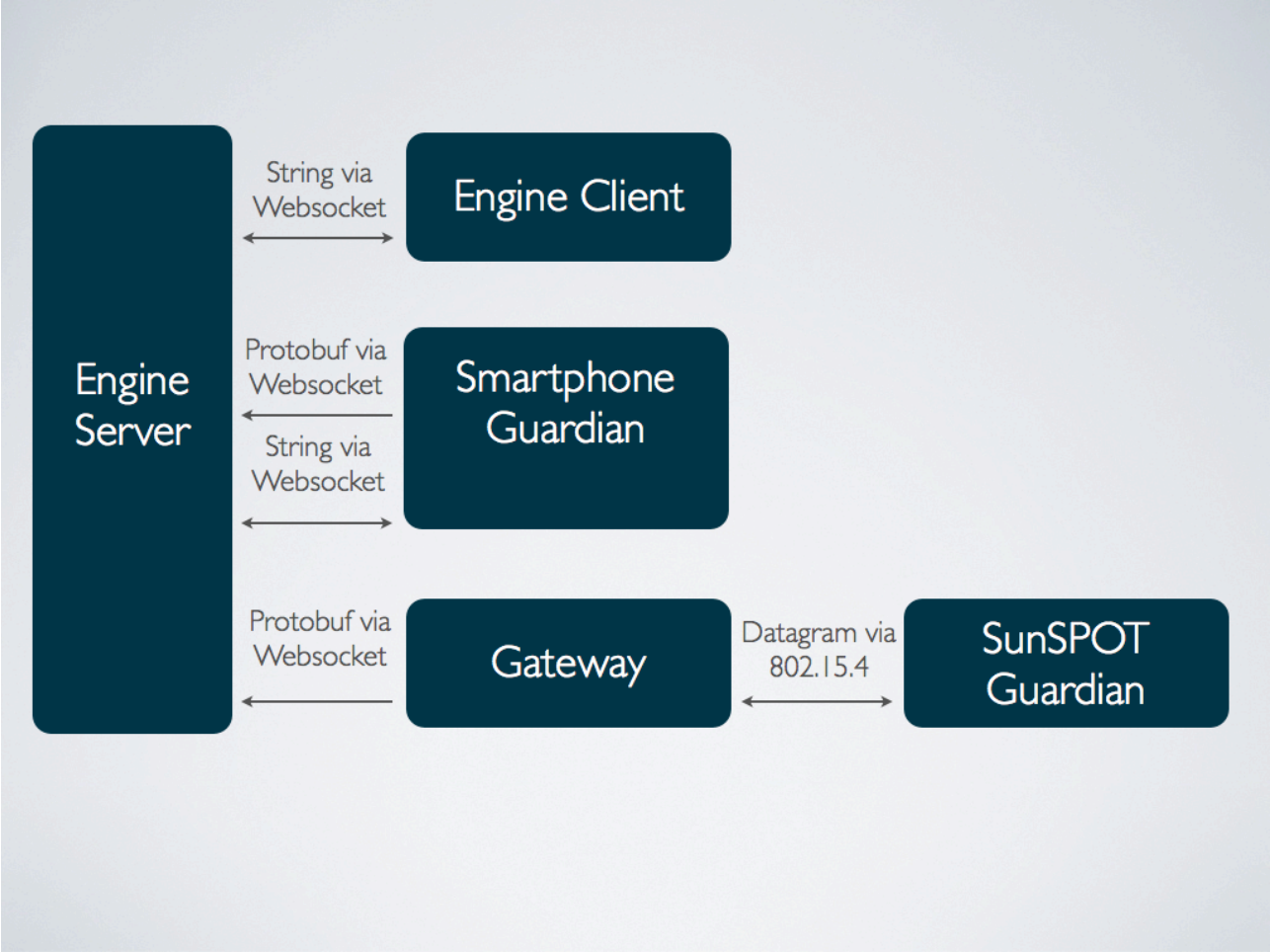
Εικόνα 5.7 Πρωτόκολλο Engine Server - Smartphone Guardian

Παρόμοια με παραπάνω, αφού εγκατασταθεί η σύνδεση ανάμεσα στα 2 επίπεδα, τότε ο Guardian πρέπει να στείλει ένα μήνυμα για να ειδοποιήσει τον Engine Server για την ταυτότητα του. Τότε ο Engine Server αποθηκεύει το αναγνωριστικό του Guardian στη λίστα που διαθέτει για όλους του Guardians.

Μόλις δημιουργηθεί ένα Event στον Guardian, τότε αφού γίνει serialized στέλνεται στον Engine Server. Αν το Event είναι αποδεκτό τότε, ανάλογα με το παιχνίδι, ο Server μπορεί να απαντήσει στον Guardian, για να δωθεί output στον παίκτη.

5.5.4 Η “μεγάλη εικόνα”

Παρακάτω ακολουθεί ένα διάγραμμα με όλα τα στοιχεία της αρχιτεκτονικής και τους τρόπους με τους οποίους αυτά επικοινωνούν, όπως περιγράφηκαν παραπάνω.



Εικόνα 5.8 Αρχιτεκτονική συστήματος και επικοινωνία υποσυστημάτων

Από το παραπάνω σχήμα εύκολα μπορεί να παρατηρηθεί το κοινό κανάλι επικοινωνίας ανάμεσα στο Engine Server και στα υπόλοιπα υποσυστήματα, που είναι τα WebSockets. Επίσης φαίνεται πως τα Events που αποστέλλονται γίνονται με χρήση των Protocol Buffers ενώ τα πρωτόκολλα διασύνδεσης ανάμεσα στα επίπεδα, και αλλαγής του Game State βασίζονται πάνω σε μηνύματα κειμένου.

Σε αυτό το σχήμα φαίνεται και η δυνατότητα του συστήματος να επεκταθεί σχετικά εύκολα σε άλλες συσκευές. Για να υποστηριχθεί λ.χ. και κάποιο άλλο

mobile λειτουργικό σύστημα, αρκεί να ακολουθεί το πρωτόκολλο ανάμεσα στα SmartPhone Guardian και Engine Server. Αν κάποιος αισθητήρας υποστηρίζει το IEEE 802.15.4, χρησιμοποιώντας το Echo Protocol, μπορεί να συνδεθεί κανονικά σε ένα Gateway.

6 Υλοποίηση Συστήματος

6.1 Tug Of War

Το πρώτο παιχνίδι που υλοποιήθηκε για την πλατφόρμα που σχεδιάστηκε είναι το Tug Of War, το οποίο είναι ένα από τα παιχνίδια που είχαν υλοποιηθεί και στην πλατφόρμα FunInNumbers με την προηγούμενη αρχιτεκτονική. Πρόκειται για ένα ανταγωνιστικό παιχνίδι, που ευνοεί την ύπαρξη πολλών παικτών ταυτόχρονα και πρόκειται ουσιαστικά για την ψηφιακή μεταφορά της Διελκυστίνδας.

6.1.1 Περιγραφή Παιχνιδιού

Οι παίκτες χωρίζονται σε 2 ομάδες. Η οθόνη του παιχνιδιού είναι χωρισμένη σε 2 επιφάνειες διαφορετικού χρώματος, με κάθε μια να αντιστοιχεί σε κάθε ομάδα. Οι παίκτες πρέπει να κάνουν γρήγορα και σωστά τις χειρονομίες που φαίνονται στην οθόνη, στη μεριά της ομάδας τους για να κερδίσει η ομάδα ένα πόντο. Για κάθε πόντο που παίρνει μια ομάδα, η επιφάνεια στην οθόνη μεγαλώνει σε βάρος της άλλης. Σκοπός κάθε ομάδας είναι να κατακτήσει χρωματικά όλη την οθόνη.

Εικόνα 6.1 Ένα παιχνίδι Tug Of War σε εξέλιξη

Οι ενδείξεις κίνησεις που μπορεί να φαίνονται στην οθόνη είναι οι εξής:

	Οριζόντια κίνηση αριστερά - δεξιά
	Κάθετη κίνηση πάνω - κάτω
	Οριζόντια κίνηση εμπρός - πίσω

ANY gesture	Οποιαδήποτε από τις παραπάνω

Πίνακας 6.2 Gestures του Tug Of War

6.1.2 Engine

Το επίπεδο Engine, το υψηλότερο επίπεδο της αρχιτεκτονικής, αποτελείται από ένα web application, για την ανάπτυξη του οποίου έχει χρησιμοποιηθεί το GWT 2.5.0, και εκτελείται σε έναν Resin v4.0.23 server.

6.1.2.1 Engine Server

Όπως ήδη περιγράφηκε στο κεφάλαιο 5, το παρόν επίπεδο χωρίζεται σε 4 υποσυστήματα. Ακολουθούν συγκεκριμένα κομμάτια κώδικα που αφορούν στην υλοποίηση του συγκεκριμένου παιχνιδιού για κάθε υποσύστημα.

- **Game Logic**

```
public static String[] currentGestures = {"TOWany", "TOWany"};
public static ArrayList<String> gestures = new ArrayList<String>
(Arrays.asList("TOWany", "TOWhor","TOWver", "TOWfor"));
public static int score[] = {0, 0};
public static int gameStatus = STATUS_NORMAL;
```

Οι παραπάνω μεταβλητές αποτελούν τις πληροφορίες που συνιστούν την τρέχουσα κατάσταση του παιχνιδού. Με τη σειρά βλέπουμε τις κινήσεις που πρέπει να κάνει η κάθε ομάδα, το σύνολο όλων των πιθανών κινήσεων, το score, καθώς και το status του παιχνιδιού.

```
public void init() {
gestureUpdateTimer.scheduleAtFixedRate(new TimerTask() {
@Override
public void run() {

TowServerLogic.currentGestures[GREEN] =
TowServerLogic.gestures.get(random.nextInt(4));

TowServerLogic.currentGestures[RED] =
TowServerLogic.gestures.get(random.nextInt(4));

TowServerLogic.getInstance().setChanged();
TowServerLogic.getInstance().notifyObservers(null);
}
}, 0, 3000);
}
```

Κάθε 3 δευτερόλεπτα, αλλάζουν τα gestures που πρέπει να κάνει η κάθε ομάδα με τυχαίο τρόπο. Με την notifyObservers ‘ξυπνάει’ ο websocket Listener, και στέλνει τις νέες κινήσεις στους browsers.

- **Java Servlet**

Αυτό το επίπεδο είναι υπεύθυνο να περιμένει πιθανά requests, και αν έρθει ένα request που είναι WebSocket το προωθεί στο υποσύστημα WebSocket Listener. Παρακάτω φαίνεται η μέθοδος service() του Servlet.

```
public void service(final ServletRequest request, final ServletResponse
response)
throws IOException, ServletException {

final HttpServletRequest req = (HttpServletRequest) request;
final HttpServletResponse res = (HttpServletResponse) response;


String protocol = req.getHeader("Sec-WebSocket-Protocol");

if (protocol != null && protocol.startsWith("TOWClient")) {
res.setHeader("Sec-WebSocket-Protocol", protocol);

} else {
res.sendError(HttpServletResponse.SC_NOT_ACCEPTABLE);
System.out.println(HttpServletResponse.SC_NOT_ACCEPTABLE);
return;
}
try {

WebSocketServletRequest wsRequest = (WebSocketServletRequest) request;
wsRequest.startWebSocket(LISTENER);


} catch (Exception e) {
e.printStackTrace();
}
}
```

Πράγματι, φαίνεται πως ελέγχεται εάν το subProtocol που χρησιμοποιείται είναι το 'TOWClient' τότε το αίτημα περνά στον Listener.

• **WebSocket Listener**

Οι πληροφορίες που χρειάζεται το υποσύστημα WebSocket Listener είναι οι εξής:

```
private final List<WebSocketContext> browserList = new
ArrayList<WebSocketContext>();
private final Map<WebSocketContext, Integer> guardianList = new
HashMap<WebSocketContext, Integer>();
private final Map<WebSocketContext, String> guardianTwitterAccounts =
new HashMap<WebSocketContext, String>();
private final Map<WebSocketContext, Integer[]> spotBridges = new
HashMap<WebSocketContext, Integer[]>();
```

```
private int[] playerCount = {0, 0};
private int[] spotCount = {0, 0};
```

Διαθέτει λίστες από τους συνδεδεμένους Browsers, τους συνδεδεμένους SmartphoneGuardians, τα Twitter accounts τους, τα συνδεδεμένα Gateways καθώς και το συνολικό αριθμό παικτών για κάθε ομάδα, αλλά και τον αριθμό παικτών με SunSPOTS.

Παρακάτω φαίνεται η onReadText του WebSocket:

```
@Override
public void onReadText(final WebSocketContext context, final Reader is)
throws IOException {

int ch;
final StringBuilder stringBuilder = new StringBuilder();
while ((ch = is.read()) >= 0) {
stringBuilder.append((char) ch);
}
is.close();

final String message = stringBuilder.toString();
final String[] words = message.split("_");


if (message.equals("ping")) {
System.out.println("Ping!");
}

else if (words[0].equals("TOWBrowser")) {
System.out.println("Browser Connected");
browserList.add(context);
sendToClient(context, "TOWGes_"
+ TowServerLogic.currentGestures[0] + "_" +
TowServerLogic.currentGestures[1]);

sendToClient(context, "TOWPlayers_"
+ String.valueOf(playerCount[RED]) + "_" +
String.valueOf(playerCount[GREEN]));
sendToClient(context, getTweepsMessage(RED));
sendToClient(context, getTweepsMessage(GREEN));

try {

if (!IP.equals("")) {
sendToClient(context, "TOWQR_" + IP);
}
```

```
} catch (Exception e) {
e.printStackTrace();
}

sendToClient(context, "TOWScore_"
+ String.valueOf(TowServerLogic.score[0]) + "_" +
String.valueOf(TowServerLogic.score[1]));

} else if (words[0].equals("TOWGuardian")) {

System.out.println("Guardian Connected");
int guardianTeam = Integer.valueOf(words[1]);
guardianList.put(context, guardianTeam);

if (guardianTeam == RED) {
playerCount[RED]++;
} else {
playerCount[GREEN]++;
}

if (!words[2].equals("-")) {

String tweep = "";
if (words.length > 3) {

tweep = message.replace(words[0] + "_" + words[1] + "_",
"").replace("_", "+");
} else {
tweep = words[2];
}

guardianTwitterAccounts.put(context, tweep);
sendToAllBrowsers(getTweepsMessage(guardianTeam));
}
sendToAllBrowsers("TOWPlayers_" + String.valueOf(playerCount[RED]) + "_"
+ String.valueOf(playerCount[GREEN]));
} else if (words[0].equals("TOWSpotBridge")) {
System.out.println("SpotBridge Connected");
spotBridges.put(context, new Integer[]{0, 0});
} else if (words[0].equals("TOWSpots")) {
System.out.println("Spots Updated");
int newSpotCount[] = {0, 0};
newSpotCount[RED] = Integer.valueOf(words[RED + 1]);
newSpotCount[GREEN] = Integer.valueOf(words[GREEN + 1]);

for (int team : new int[]{GREEN, RED}) {
playerCount[team] -= spotBridges.get(context)[team];
playerCount[team] += newSpotCount[team];
}
```

```
spotBridges.remove(context);
spotBridges.put(context, new Integer[]{newSpotCount[RED],
newSpotCount[GREEN]});
sendToAllBrowsers("TOWPlayers_" + String.valueOf(playerCount[RED]) + "_"
+ String.valueOf(playerCount[GREEN]));

} else if (words[0].equals("TOWSpotBridge")) {
System.out.println("SpotBridge Connected");
spotBridges.put(context, new Integer[]{0, 0});
} else if (words[0].equals("TOWSpots")) {

System.out.println("Spots Updated");
int newSpotCount[] = {0, 0};
newSpotCount[RED] = Integer.valueOf(words[RED + 1]);
newSpotCount[GREEN] = Integer.valueOf(words[GREEN + 1]);

for (int team : new int[]{GREEN, RED}) {
playerCount[team] -= spotBridges.get(context)[team];
playerCount[team] += newSpotCount[team];
}

spotBridges.remove(context);
spotBridges.put(context, new Integer[]{newSpotCount[RED],
newSpotCount[GREEN]});
sendToAllBrowsers("TOWPlayers_" + String.valueOf(playerCount[RED]) + "_"
+ String.valueOf(playerCount[GREEN]));

}
}
```

Η παραπάνω συνάρτηση διαχειρίζεται τις εξής περιπτώσεις:

I. Ping message: Οι συνδέσεις WebSocket δεν χαρακτηρίζονται από keep alive πολιτική. Άρα ο προγραμματιστής οφείλει να κρατάει ανοιχτή τη σύνδεση υλοποιώντας ping-pong μηνύματα. Ο server λοιπόν δέχεται ping μηνύματα.

II. Σύνδεση ενός Engine Client (Browser): Όπως αναφέραμε παραπάνω, σύμφωνα με το πρωτόκολλο σύνδεσης του Engine Server και του Engine Client, μόλις ένας client συνδεθεί, στέλνει μήνυμα στον server, και ο server τον αποθηκεύει και ύστερα στέλνει την τρέχουσα κατάσταση του παιχνιδιού. Στην προκειμένη περίπτωση αποστέλλονται οι κινήσεις που πρέπει να κάνουν οι ομάδες, ο αριθμός των παικτών για κάθε ομάδα, μαζί

με τα Twitter Accounts, αν υπάρχουν, η IP διεύθυνση του Server, και τέλος το τρέχον score.

III. Σύνδεση ενός SmartphoneGuardian: Αποθηκεύεται ο νέος αριθμός παικτών, και ενημερώνονται όλοι οι Clients.

IV. Σύνδεση ενός Gateway: Αποθηκεύεται το Gateway από τον server.

V. Αλλαγή αριθμού παικτών με SunSPOTs: Αποθηκεύεται ο νέος αριθμός παικτών με SPOTs και στέλνεται σε όλους του Browsers για να οπτικοποιηθεί στην οθόνη.

Παρακάτω φαίνονται οι συναρτήσεις που εκτελούνται όταν ο server δεχθεί μήνυμα που αφορά σε κίνηση παίκτη:

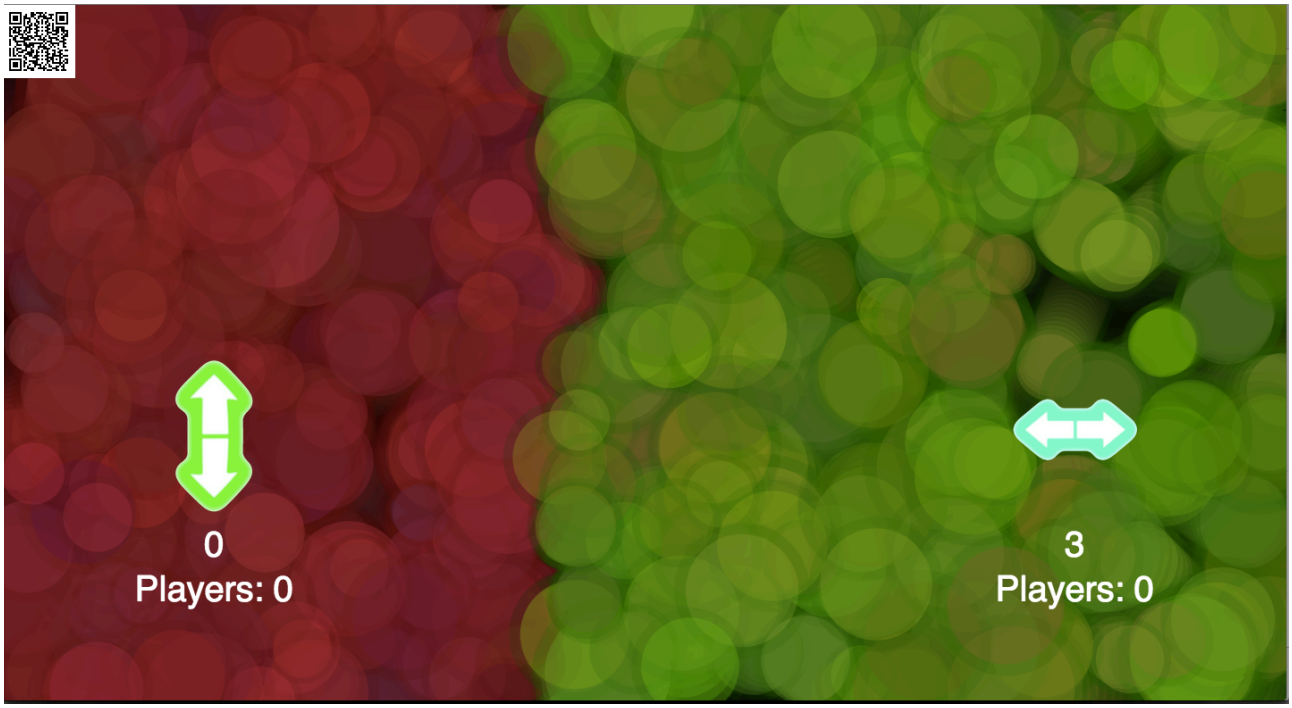
```
public void onReadBinary(com.caucho.websocket.WebSocketContext context,
java.io.InputStream is) {
System.out.println("onReadBinary");

Message.ProtoEvent event = generateProtoEventFromInputStream(is);
System.out.println(event);
if (event == null) {
return;
}

if (event.getType().equals("TOW")) {
processGestureMessage(context, event);
}
}

public Message.ProtoEvent
generateProtoEventFromInputStream(java.io.InputStream is) {
try {


final byte[] bytes = new byte[1024];
int numberOfBytes = is.read(bytes);
final byte[] clearData = new byte[numberOfBytes];
System.arraycopy(bytes, 0, clearData, 0, numberOfBytes);
final ByteString byteString = ByteString.copyFrom(clearData);
return (Message.ProtoEvent.parseFrom(byteString));
} catch (Exception e) {
e.printStackTrace();
return null;
}
}
```

```
public void processGestureMessage(WebSocketContext guardian,
Message.ProtoEvent event) {

if (guardianList.containsKey(guardian) && guardianList.get(guardian) !=
event.getTeamId()) {
guardianChangedTeam(guardian);
}


if (TowServerLogic.gameStatus == STATUS_NORMAL &&
(TowServerLogic.currentGestures[event.getTeamId()].equals("TOWany")
||
event.getDescription().equals(TowServerLogic.currentGestures[event.getTe
amId()]))) {
TowServerLogic.score[event.getTeamId()]++;
if (guardianList.containsKey(guardian)) {
sendToClient(guardian, "TOWack");
}
if (Math.abs(TowServerLogic.score[0] - TowServerLogic.score[1]) >
20 - ((TowServerLogic.score[0] - TowServerLogic.score[1]) >= 0 ?
TowServerLogic.score[0] : TowServerLogic.score[1]) / 20) {

endGame(TowServerLogic.score[RED] > TowServerLogic.score[GREEN] ? RED :
GREEN);


} else {
sendToAllBrowsers("TOWScore_" + String.valueOf(TowServerLogic.score[0])
+ "_" + String.valueOf(TowServerLogic.score[1]));
}
}
}
```

Τότε εκτελείται η συνάρτηση onReadBinary του WebSocket. Στην περίπτωση μηνύματος κίνησης από παίκτη, ο server λαμβάνει δεδομένα που έχουν γίνει serialized με βάση το αρχείο message.proto. Τότε λοιπόν με την συνάρτηση generateProtoEventFromInputStream γίνεται το deserialization, και τελικά με τη processGestureMessage, ο WebSocket Listener επεξεργάζεται το μήνυμα. Αν η κίνηση που έχει κάνει ο παίκτης είναι σωστή, του απαντά με ένα ACK μήνυμα, και ύστερα ενημερώνει την τρέχουσα κατάσταση του παιχνιδιού. Έπειτα στέλνεται μήνυμα σε όλους του Clients για να οπτικοποιήσουν τη νέα κατάσταση.

6.1.2.2 Engine Client

Όπως αναλύθηκε παραπάνω το συγκεκριμένο επίπεδο είναι υπεύθυνο για την οπτικοποίηση του παιχνιδιού και αποτελείται από 4 υποσυστήματα. Παρακάτω φαίνεται μια εικόνα από την εκτέλεση του client σε ένα περιηγητή ιστού.



Εικόνα 6.3 Το 'ταμπλώ' του Tug Of War σε ένα Περιηγητή Ιστού

Βλέπουμε πως η οθόνη είναι χωρισμένη σε 2 επιφάνειες, την κόκκινη και την πράσινη, με κάθε μια από αυτές να αναπαριστά το χώρο που "πιάνει" μια ομάδα. Αριστερά και δεξιά φαίνονται οι χειρονομίες που πρέπει να κάνει κάθε ομάδα, το score και ο αριθμός των παικτών σε κάθε ομάδα αυτή τη στιγμή.

- **HTML5**

Τα στοιχεία που συνθέτουν την παραπάνω εικόνα, φαίνονται στο αρχείο html που περιγράφει την εφαρμογή.

```
<canvas id="towCanvas" style="position:absolute; top:0; left:0;" data-
processing-sources="Circles.pde" ></canvas>

<div id="leftScoreDiv" style="position:absolute; top:50%;left:10% "></
div>
```

```
<div id="rightScoreDiv" style="position:absolute; top:50%;right:10% "></
div>


<div id="leftTweepsDiv" class="twitterBar twitterBarLeft" ></div>
<div id="rightTweepsDiv" class="twitterBar twitterBarRight" ></div>
```

Το στοιχείο HTML5 canvas είναι αυτό που απεικονίζει το sketch της processing.js με τους κινούμενους κύκλους που φαίνεται στην εικόνα. Τα 4 στοιχεία div που ακολουθούν είναι στοιχεία τα οποία χρησιμοποιεί το GWT υποσύστημα του Engine Client βάζοντας διάφορα widgets για να οπτικοποιηθούν το score των ομάδων και τα twitter accounts των παικτών.

- **Processing.js sketch**

Παρακάτω φαίνονται οι βασικές συναρτήσεις ενός processing sketch, η setup() που καλείται μια φορά στην αρχή, και η draw, που καλείται συνεχόμενα.

```
final ArrayList<Circle>[] mole = new ArrayList[2];
int gameStatus = STATUS_NORMAL;
float alpha = 0;


void setup() {
    size(window.innerWidth, window.innerHeight);
    background(0);
    frameRate(30);
    for (int surface = LEFT; surface <= RIGHT; surface++) {
        mole[surface] = new ArrayList<Circle>();
        for (int i = 0; i < 200; i++) {
            mole[surface].add(new
Circle(int(random(width)),int(random(height)), surface));
            }
     }
}


void draw() {
     if (gameStatus == STATUS_RESTARTING) {
            eraser(color(255), 255);
            gameStatus = STATUS_NORMAL;
            }
   if (gameStatus == STATUS_NORMAL) {
            redrawMorphome();
            }
```

```
else if (gameStatus == STATUS_WINNING && gameWinner == GREEN) {
            eraser(color(0, 100, 0), alpha)
            alpha = alpha + 0.06;

            }
else if (gameStatus == STATUS_WINNING && gameWinner == RED) {
            eraser(color(100, 0, 0), alpha)
            alpha = alpha + 0.06;
            }
            }
```

Η setup() δημιουργεί λίστες από κύκλους, 1 για κάθε ομάδα, και τις τοποθετεί στον επίπεδο. Η draw() ανάλογα με την τρέχουσα κατάσταση του παιχνιδιού ‘ζωγραφίζει’ στην οθόνη το κατάλληλο οπτικό αποτέλεσμα. Στην περίπτωση που το παιχνίδι συνεχίζεται κανονικά εκτελούνται τα εξής:

```
void redrawMorphome() {
eraser(color(0), 20);
displayCircles();
}

void displayCircles() {
for (Circle circle : mole[RED]) {
circle.move();
circle.display();
}

for (Circle circle : mole[GREEN]) {
circle.move();
circle.display();
}

}

void eraser(final color c, final int alpha) {
fill(c, alpha);
noStroke();
rect(0, 0, width, height);
}
```

Επίσης, όταν κάποια ομάδα πετύχει κάποιο πόντο, τότε εκτελείται μία απο τις παρακάτω συναρτήσεις:

```
void doRightMovement(final int ratio) {
for (Circle circle : mole[RED]) {
circle.expandRight(ratio);
```

```
}

for (Circle circle : mole[GREEN]) {
circle.shrinkLeft(ratio);
}

}

void doLeftMovement(final int ratio) {
for (Circle circle : mole[RED]) {
circle.shrinkRight(ratio);
}

for (Circle circle : mole[GREEN]) {
circle.expandLeft(ratio);
}

}
```

- **Javascript**

Η Javascript στη συγκεκριμένη εφαρμογή, είναι κυρίως υπεύθυνη για τον κύκλο ζωής των websockets, και είναι το μέσο που επιτρέπει στο GWT class να επικοινωνεί με το Processing.js Sketch. Όταν ξεκινά η εκτέλεση του προγράμματος εκτελείται η παρακάτω συνάρτηση.

```
function init()
{
ws = new WebSocket(url, "TOWClient");
ws.onopen = wsopen;
ws.onmessage = wsmessage;
ws.onclose = wsclose;
window.onresize = function(){
canvas = document.getElementById("towCanvas");
canvas.width = document.width;
canvas.height = document.height;
getProcessingInstance().resize();
}
}
```

Σε αυτή τη συνάρτηση αρχικοποιείται το WebSocket, και το canvas element μέσα στο οποίο η Processing.js θα σχηματίσει τα γραφικά του παιχνιδιού. Οι βασικές συναρτήσεις που αφορούν στο WebSocket φαίνονται παρακάτω:

```
function wsopen(event) {
```

```
    try {

        sendMessage("TOWBrowser");

        myVar = setInterval(function () {
                            sendPing()
                            }, 6000);

    } catch (e) {
        alert("error while sending message : " + e);
    }
}

function wsmessage(event) {
    try {


        console.log(event.data);

        window.handleWebSocketMessage(event.data);

    } catch (e) {
        console.log("on message : " + e);
    }
}

function wsclose(event) {
    console.log("Shouldn't be printed!!");
    clearInterval(myVar);

    ws = new WebSocket(url, "TOWClient");
    ws.onopen = wsopen;
    ws.onmessage = wsmessage;
    ws.onclose = wsclose;
}


function sendPing() {
    ws.send("ping");
}

function sendMessage(message){
    ws.send(message);
}
```

- **GWT Class**

Η GWT Class είναι υπεύθυνη για τον συντονισμό και την εκτέλεση της εφαρμογής. Παρακάτω φαίνεται η συνάρτηση onModuleLoad που εκτελείται μόλις ξεκινήσει να εκτελείται η εφαρμογή.

```
public void onModuleLoad() {

    startSound.play();
    exportHandleWebSocketMessage(this);

    initTeams();
    loadGestureFrames();
    RootPanel.get("leftScoreDiv").add(assembleScoreTable(RED));
    RootPanel.get("rightScoreDiv").add(assembleScoreTable(GREEN));
    RootPanel.get("leftTweepsDiv").add(tweepsTable[RED]);
    RootPanel.get("rightTweepsDiv").add(tweepsTable[GREEN]);


    if (refreshGestureImagesTimer == null) {
        refreshGestureImagesTimer = new Timer() {
            @Override
            public void run() {


gestureImages[RED].setUrl(imagesForGestures.get(currentGestures[RED]).ge
t(currentGestureImageFrame));

gestureImages[GREEN].setUrl(imagesForGestures.get(currentGestures[GREEN]
).get(currentGestureImageFrame));
                currentGestureImageFrame++;

                if (currentGestureImageFrame == 8) {
                    currentGestureImageFrame = 1;
                }


            }
        };
    }

refreshGestureImagesTimer.scheduleRepeating(GESTURE_IMAGE_REFRESH_RATE);
    RootPanel.get().addDomHandler(keyDownHandler,
KeyDownEvent.getType());
    initJSNI();
}
```

Κατα την εκτέλεση αυτής της συνάρτησης εισάγονται τα gadgets στα divs της html που αναφέρθηκαν παραπάνω, ορίζονται 2 timers για να κινούνται οι εικόνες που θα υποδεικνύουν στους παίκτες το είδος της κίνησης που πρέπει να κάνουν, και μέσω του JSNI καλείται η init της javascript που φαίνεται παραπάνω.

Η συνάρτηση που είναι υπεύθυνη να επεξεργάζεται τα μηνύματα που έρχονται από το WebSocket είναι η εξής:

```
public void handleWebSocketMessage(String msg) {

    String[] words = msg.split("_");

    if (words[0].equals("TOWGes")) {
        currentGestures[RED] = words[1 + RED];
        currentGestures[GREEN] = words[1 + GREEN];
        currentGestureImageFrame = 0;
    } else if (words[0].equals("TOWScore")) {

        int redS = Integer.valueOf(words[1 + RED]);
        int greenS = Integer.valueOf(words[1 + GREEN]);

        if (redS > Integer.valueOf(scoreLabels[RED].getText())) {
            for (int i = Integer.valueOf(scoreLabels[RED].getText()) +
1; i <= redS; i++) {
                goRedJSNI(INC_SIMPLE + i / 10);

            }
        }
        if (greenS > Integer.valueOf(scoreLabels[GREEN].getText())) {
            for (int i = Integer.valueOf(scoreLabels[GREEN].getText()) +
1; i <= greenS; i++) {
                goGreenJSNI(INC_SIMPLE + i / 10);
            }


        }

        scoreLabels[RED].setText(words[1 + RED]);
        scoreLabels[GREEN].setText(words[1 + GREEN]);

    } else if (words[0].equals("TOWWin")) {
        winSound.play();
        win(Integer.valueOf(words[1]));
    } else if (words[0].equals("TOWRestart")) {
        startSound.play();
```

```
        restartGame();

    } else if (words[0].equals("TOWPlayers")) {
        playerCountLabels[RED].setText("Players: " + words[1 + RED]);
        playerCountLabels[GREEN].setText("Players: " + words[1 +
GREEN]);
    } else if (words[0].equals("TOWQR")) {

        generateQRJSNI(words[1]);
    } else if (words[0].equals("TOWTweeps")) {
        int team = Integer.valueOf(words[1]);
        ArrayList<String> tweeps = new ArrayList<String>();

        for (int i = 2; i < words.length; i++) {

            tweeps.add(words[i]);
        }

        updateTweepsTable(team, tweeps);

    }
}
```

Η συνάρτηση αυτή επεξεργάζεται τις εξής περιπτώσεις:

- Νέες κινήσεις για τις ομάδες
- Νέο score
- Νίκη μιας ομάδας
- Επανεκκίνηση του παιχνιδιού
- Νέος αριθμός παικτών
- QR Code της IP διεύθυνσης του Server
- Twitter accounts των παικτών

Σε κάθε περίπτωση καλείται μια από τις παρακάτω συναρτήσεις η οποία ανανεώνει το περιεχόμενο του παραθύρου του Browser:

```
public void win(int team) {

    winJSNI(team);
    refreshGestureImagesTimer.cancel();
    for (int i = 0; i <= 1; i++) {
        playerCountLabels[i].setVisible(false);
        gestureImages[i].setUrl("img/null.gif");
    }
```

```
}


public void restartGame() {
    for (int t = 0; t <= 1; t++) {
        scoreLabels[t].setText(String.valueOf(0));
        playerCountLabels[t].setVisible(true);
        scoreLabels[t].setVisible(true);
        gestureImages[t].setVisible(true);
    }

    restartJSNI();

refreshGestureImagesTimer.scheduleRepeating(GESTURE_IMAGE_REFRESH_RATE);

}
public void updateTweepsTable(int team, ArrayList<String> tweeps) {

    tweepsTable[team].removeAllRows();
    int counter = 0;
    for (final String tweep : tweeps) {

        final String finalTweep = tweep.replace("+", "_");
        Image tweepIcon = new Image();
        tweepIcon.setTitle("@" + finalTweep);
        tweepIcon.setUrl("https://api.twitter.com/1/users/
profile_image/" + finalTweep);
        tweepIcon.addClickHandler(new ClickHandler() {
                                    @Override
                                    public void onClick(ClickEvent event)
{

                                    Window.open("http://www.twitter.com/"
+ finalTweep, "_blank", "");
                                    }
                                    });

        tweepsTable[team].setWidget(counter, 0, tweepIcon);
        counter++;
    }

}
```

Οι συναρτήσεις που αποτελούν τη ‘γέφυρα’ μεταξύ του GWT και τις Processing.js χρησιμοποιούν το JSNI καλώντας γνήσια Javascript, και φαίνονται παρακάτω:

```
public native void winJSNI(int team)
/*-{
$wnd.getProcessingInstance().win(team);
}-*/;

public native void restartJSNI()
/*-{
$wnd.getProcessingInstance().restartGame();
}-*/;


private native static void goRedJSNI(final int ratio)
/*-{
$wnd.getProcessingInstance().doRightMovement(ratio);
}-*/;

private native static void goGreenJSNI(final int ratio)
/*-{
$wnd.getProcessingInstance().doLeftMovement(ratio);
}-*/;


public native void generateQRJSNI(String IP)
/*-{
$wnd.generateQR(IP);
}-*/;

public native void exportHandleWebSocketMessage(TugOfWar thisTow)
/*-{
$wnd.handleWebSocketMessage = function (msg) {
return
thisTow.@eu.funinnumbers.gwt.tugofwar.client.TugOfWar::handleWebSocketMe
ssage(Ljava/lang/String;)(msg);
}
}-*/;


public native void sendWebSocketMessageJSNI(String s)
/*-{
$wnd.sendMessage(s);
}-*/;


private native static void initJSNI()
/*-{
$wnd.init();
}-*/;
```

6.1.3 Gateway

Το Gateway επίπεδο είναι το επίπεδο που ενώνει τους SPOT Guardians με το Engine. Η διασύνδεση με τους SPOT Guardians γίνεται με χρήση του Echo Protocol, πρωτοκόλλου που δημιουργήθηκε στην προηγούμενη έκδοση της πλατφόρμας FunInNumbers και περιγράφεται στο **[13]**.

Η σύνδεση με τον Server γίνεται με WebSockets. Παρακάτω φαίνονται οι μέθοδοι που συνδέουν και αποσυνδέουν σωστά το Gateway με τον server.

```
private WebSocketClientFactory factory;
private WebSocketClient client;
private WebSocket.Connection connection;
private URI WS_URI;

public void tryToConnect(){

    try {
        Logger.getInstance().debug("Trying to connect to server...");
        WS_URI = new URI(TOWWSURL);
        factory = new WebSocketClientFactory();
        factory.setBufferSize(4096);
        factory.start();
        webSocketImpl = new WebSocketImpl();
        client = factory.newWebSocketClient();
        client.setMaxIdleTime(-1);
        client.setMaxBinaryMessageSize(1024);
        client.setProtocol("TOWClient");
        connection = client.open(WS_URI, webSocketImpl).get();


        pingTimer = new Timer();
        pingTimer.scheduleAtFixedRate(new TimerTask() {
                                        @Override
                                        public void run() {
                                        try {

WSEventProcessor.getInstance().getConnection().sendMessage("ping");
```

```
                                        } catch (IOException e) {
                                        Logger.getInstance().debug("Cannot
ping Server!",e);
                                        }
                                        }
                                        }, 0, 10000);



        connection.sendMessage("TOWSpotBridge");



    } catch (Exception e) {
        Logger.getInstance().debug("Cannot Reconnect!",e);
    }
}

public void disconnect() {

    if (factory.isRunning()) {
        factory.destroy();
    }
    connection.disconnect();

    if(pingTimer!=null)
    {
        pingTimer.cancel();
    }

    try {
        factory.stop();
    } catch (final Exception e) {
        Logger.getInstance().debug("Unable to stop WebSocket Factory ",
e);
    }
}
```

6.1.4 Guardian

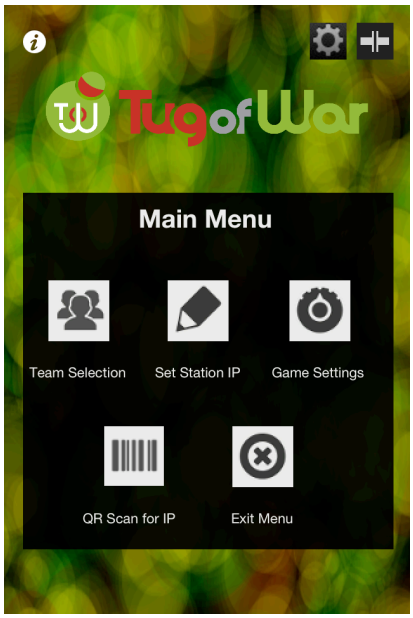
6.1.4.1 SunSPOT Guardian

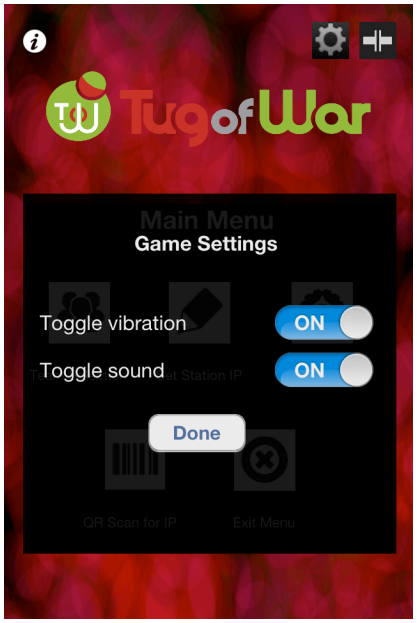
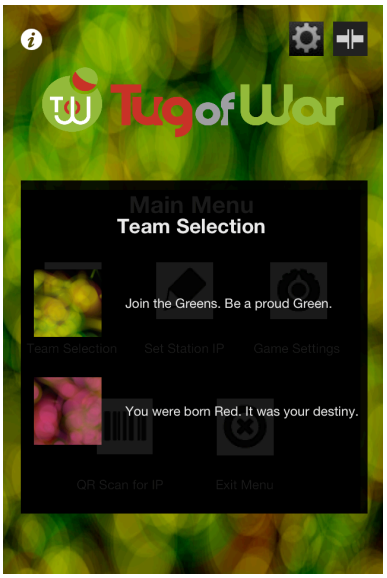
Όπως και σε οποιοδήποτε παιχνίδι υλοποιημένο και στην προηγούμενη πλατφόρμα, έτσι και εδώ, το λογισμικό που εκτελείται στο επίπεδο SunSPOT Guardian είναι ακριβώς το ίδιο.

6.1.4.2 iOS Guardian

- **View**

Το View της εφαρμογής για iOS φαίνεται στις παρακάτω εικόνες

	Βασική οθόνη του παιχνιδιού.
	Βασικό Menu επιλογών.

	Μενου με επιλογές ήχου και δόνησης. Κάθε φορά που ο παίκτης κάνει σωστή κίνηση, ειδοποιείται με δόνηση και/ήχο. Από αυτό το menu αυτό μπορεί να απενεργοποιηθεί.
	Μενου επιλογής ομάδας.

	Κάθε φορά που κάποιος κάνει μια κίνηση, και αυτή ανιχνεύεται από το τηλέφωνο, τότε επιδεικνύεται στην οθόνη.

Πίνακας 6.4 Εικόνες της iOS εφαρμογής του Tug Of War

Το Interface της εφαρμογής με το χρήστη είναι πολύ απλό. Υπάρχει ένα βασικό View, το background του οποίου δηλώνει την ομάδα στην οποία ανήκει ο παίκτης. Επίσης υπάρχει ένα menu με βασικές λειτουργίες, όπως Επιλογή Ομάδας, Ρυθμίσεις Δόνησης και Ήχου και Εισαγωγή IP διεύθυνσης του Server, είτε με κείμενο είτε με scanning ενός QR code.

• View Controller

Ο View Controller είναι η κλάση η οποία ευθύνεται για τη μορφοποίηση του View, ανάλογα με τις πράξεις που κάνει ο χρήστης. Για παράδειγμα, όταν ο χρήστης πατήσει το κουμπί με το γρανάζι που φαίνεται στην πάνω δεξιά γωνία της οθόνης, καλείται η εξής συνάρτηση.

```
- (IBAction)showMainMenuView
{
    if(self.mainMenuPane.alpha==0)
    {
        [UIView animateWithDuration:0.4 animations:^{
            self.mainMenuPane.alpha = 1;
        }];
    }
    else
    {
        [UIView animateWithDuration:0.4 animations:^{
            self.mainMenuPane.alpha = 0;
        }];
```

```
    }
    self.gestureImage.image = nil;
}
```

Το main menu είναι “ζωγραφισμένο” πάνω σε ένα Panel, το οποίο αρχικά είναι κρυφό. Αυτή η συνάρτηση το φανερώνει αν η τρέχουσα κατάσταση του είναι hidden και αντίστροφα. Υπάρχουν αντίστοιχες συναρτήσεις που καλούνται για κάθε επιλογή του χρήστη.

• Gesture Recognizer

Το υποσύστημα Gesture Recognizer είναι εκείνο που ανιχνεύει τα Gestures που κάνει ο χρήστης με τη συσκευή του. Η ανίχνευση των κινήσεων γίνεται με την παρακάτω συνάρτηση.

```
- (void)motionRecognizer:(NSTimer *) timer
{
    if([self.motionManager isDeviceMotionActive])
    {
        CMDeviceMotion *data = self.motionManager.deviceMotion;
        double sumX=0,sumY=0,sumZ=0;
        double x=0, y=0, z=0;

        x=data.userAcceleration.x;
        y=data.userAcceleration.y;
        z=data.userAcceleration.z;

        //Reads the userAcceleration Data, and keeps only values that
are bigger than 1.8.

        if(fabs(x)>1.8 || fabs(y)>1.8 || fabs(z)>1.8)
        {
            if(fabs(x)>1.8){
                y=0;
                z=0;
            }

            else if(fabs(y)>1.8){
                x=0;
                z=0;
            }
            else if(fabs(z)>1.8){
                y=0;
                x=0;
            }
```

```
        }
        else
        {
            x=0;
            y=0;
            z=0;
        }

        self.accel_x[self.sampleIndex % 4] = fabs(x);
        self.accel_y[self.sampleIndex % 4] = fabs(y);
        self.accel_z[self.sampleIndex % 4] = fabs(z);

        self.sampleIndex++;

        //Sums.

        for (int i=0;i<4;i++)
        {
            sumX+=self.accel_x[i];
            sumY+=self.accel_y[i];
            sumZ+=self.accel_z[i];
        }

        //Depending on the sums, finds which movement user did, and
calls the appropriate movement method.


        if (sumX>3.6)
        {
            //Horizontal Movement
            if((double)CFAbsoluteTimeGetCurrent() -
self.lastGestureTimeStamp > 0.18)
            {
                NSLog(@"Horizontal");
                self.lastGestureTimeStamp =
(double)CFAbsoluteTimeGetCurrent();
                [[self viewController]  doHorizontalMovement];
                [self doBufferCleanup];
                            }

        }

        else if (sumZ>2)
        {
            //Vertical Movement
            if((double)CFAbsoluteTimeGetCurrent() -
self.lastGestureTimeStamp > 0.18)
            {
                NSLog(@"Vertical");
```

```
                self.lastGestureTimeStamp =
(double)CFAbsoluteTimeGetCurrent();
                [[self viewController] doVerticalMovement];
                [self doBufferCleanup];

            }

        }

        else if (sumY>3.6)
        {
            //Forward Movement
            if((double)CFAbsoluteTimeGetCurrent() -
self.lastGestureTimeStamp > 0.18)
            {
                NSLog(@"Forward");
                self.lastGestureTimeStamp =
(double)CFAbsoluteTimeGetCurrent();
                    [[self viewController] doForwardMovement];
                [self doBufferCleanup];
                  }
        }
    }
}
```

Η συνάρτηση αυτή καλείται κάθε 50ms. Αυτό που κάνει είναι να αποθηκεύει 4 συνεχόμενες τιμές του CMMotionManager και να τις αθροίζει. Αν η συνολική τιμή ξεπερνά ένα προκαθορισμένο όριο για κάθε άξονα, τότε ανιχνεύεται μια κίνηση. Τα προκαθορισμένα αυτά όρια προέκυψαν από πειράματα που έγιναν έτσι ώστε να είναι όσο το δυνατόν πιο ακριβή.

• WebSocket Manager

Αυτό το service είναι υπεύθυνο για την υλοποίηση του WebSocket client, καθώς και των μεθόδων που έχουν να κάνουν με το Serialization και την αποστολή των δεδομένων στον Server. Παρακάτω φαίνεται ο κώδικας της onOpen() στον οποίο φαίνεται η διατήρηση του πρωτοκόλλου επικοινωνίας Smartphone Client - Engine Server που περιγράφηκε παραπάνω.

```
- (void)webSocketDidOpen:(SRWebSocket *)webSocket
{

    NSLog(@"Did Open");
    TOWViewController *vc = [self viewController];
```

```
    self.pingTimer = [NSTimer scheduledTimerWithTimeInterval:6
target:self selector:@selector(webSocketPing) userInfo:nil repeats:YES];
    self.isOpen = true;
    NSString *message = @"TOWGuardian_";
    message = [message stringByAppendingFormat:@"%d",vc.team];
    message = [message stringByAppendingString:@"_"];
    [self webSocketSendText:[message
stringByAppendingString:vc.twitterUsername]];

    vc.connectedImage.image = [UIImage imageNamed:@"Connected.png"];
}
```

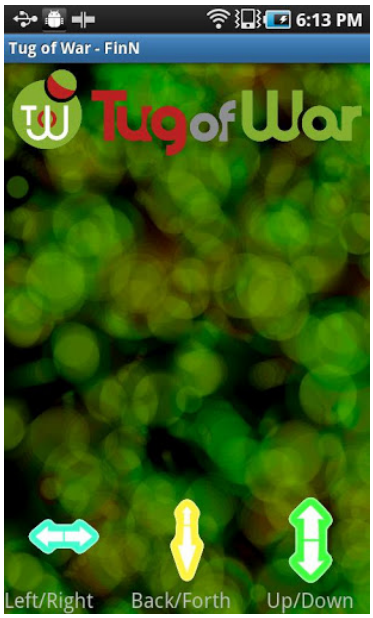
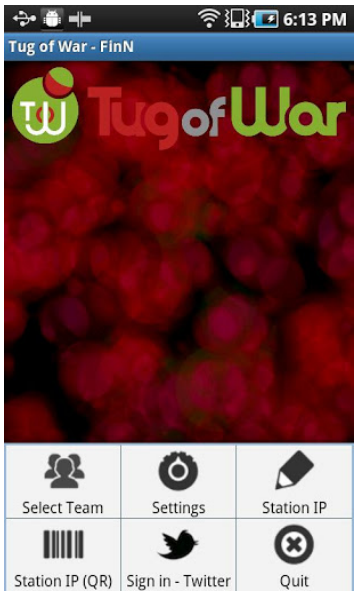
Μόλις ανοίξει το WebSocket, ο Guardian στέλνει ένα μήνυμα που ενημερώνει ότι είναι Guardian (το μήνυμα ξεκινά με το String “TOWGuardian”). Όταν ανιχνευθεί μια κίνηση και προκύψει ένα event τότε καλούνται οι εξής συναρτήσεις:

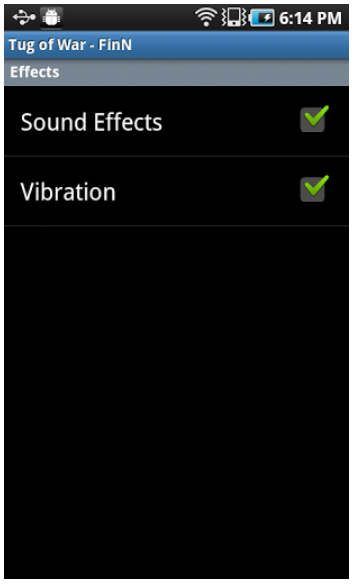
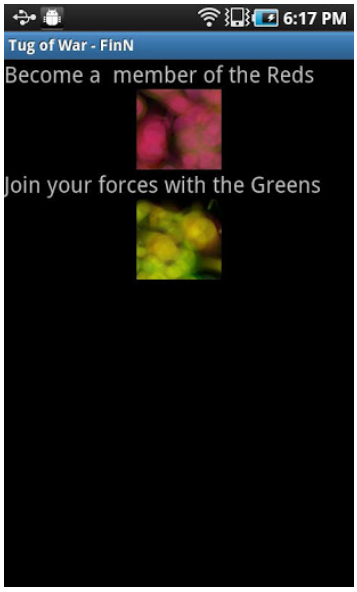
```
//Generates ProtoEvent (Google Protocol Buffer).

-(void) generateMessage: (NSString *) description team:(int) teamId
forEvent:(ProtoEvent *)event
{

    event->set_id(1);
    event->set_guardiancounter(1);
    event->set_teamid(teamId);
    event->set_type("TOW");
    event->set_description([description UTF8String]);
    event->set_battleengine(1);
    event->set_timestamp((long)CFAbsoluteTimeGetCurrent());
    event->set_stationid(1);

}

//Sends ProtoEvent to Sever.

- (void) webSocketSendEventWithDescription: (NSString*) description
andTeamId: (int) teamId
{
    ProtoEvent *toSend =new ProtoEvent;
    [self generateMessage:description team:teamId forEvent:toSend];
    std::string ps = toSend->SerializeAsString();
    [self.ws send: [NSData dataWithBytes:ps.c_str() length:ps.size()]];
    delete toSend;

}
```

Παρατηρούμε τη μέθοδο η οποία κάνει Serialize τα δεδομένα σύμφωνα με το ProtoEvent που έχει περιγραφεί παραπάνω.

6.1.4.3 Android Guardian

Αντίστοιχα με τον iOS Guardian ακολουθούν εικόνες που δείχνουν το Interface της εφαρμογής με το χρήστη.

	Βασική οθόνη του παιχνιδιού.
	Βασικό Menu επιλογών.

	Μενου με επιλογές ήχου και δόνησης. Κάθε φορά που ο παίκτης κάνει σωστή κίνηση, ειδοποιείται με δόνηση και/ήχο. Από αυτό το menu αυτό μπορεί να απενεργοποιηθεί.
	Μενου επιλογής ομάδας.

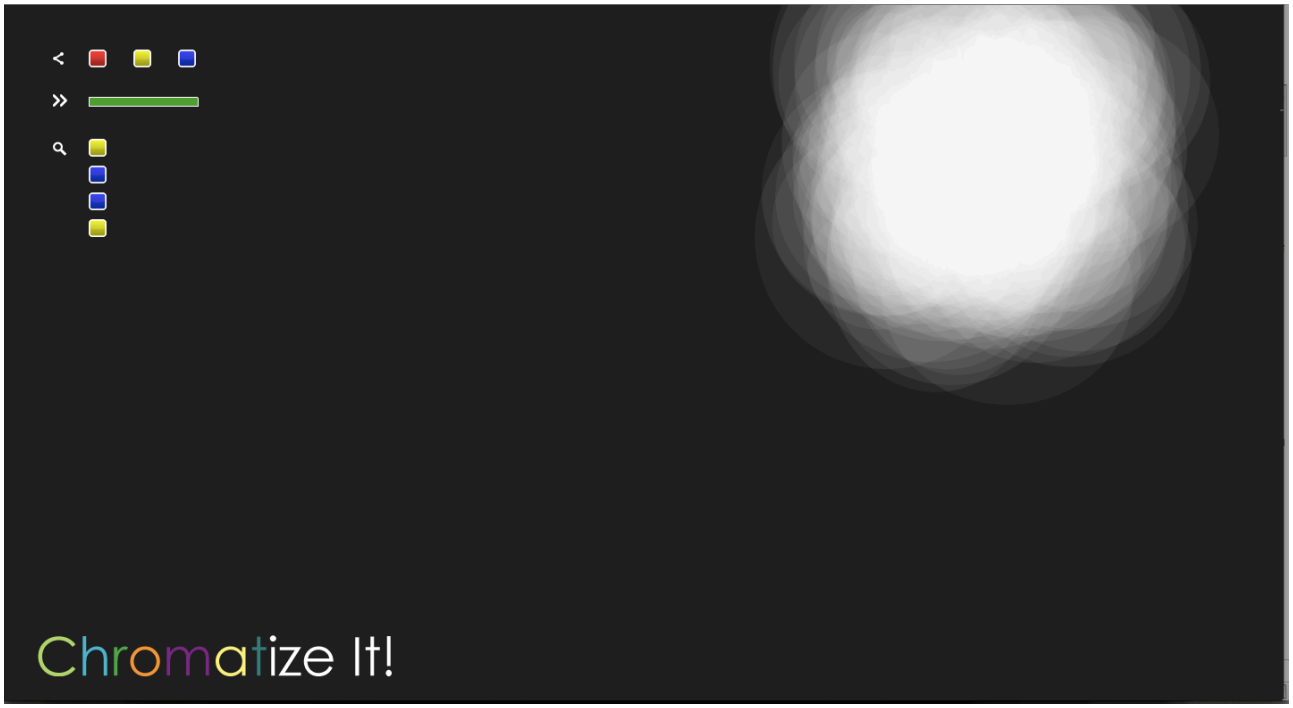
Πίνακας 6.5 Εικόνες της Android εφαρμογής του Tug Of War

6.2 Chromatize It!

Το δεύτερο παιχνίδι που υλοποιήθηκε είναι το Chromatize It!. Και αυτό το παιχνίδι είχε υλοποιηθεί στην προηγούμενη αρχιτεκτονική, και πρόκειται περισσότερο για ένα παιχνίδι περισσότερο για μικρά παιδιά.

6.2.1 Περιγραφή παιχνιδιού

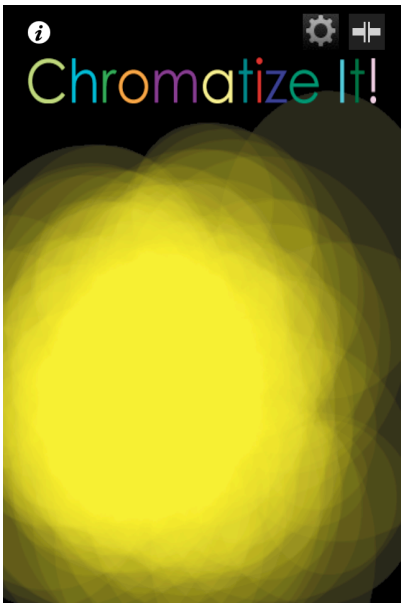
Σε αυτό το παιχνίδι οι παίκτες πειραματίζονται με την ανάμιξη των χρωμάτων. Έχουν στη διάθεση τους 3 χρώματα, κόκκινο, μπλέ και κίτρινο, και σκοπός είναι να χρωματίσουν μια αρχικά λευκή μάζα με το ζητούμενο χρώμα, “πετώντας” το χρώμα που έχουν στη συσκευή τους στην οθόνη με μια απότομη κίνηση. Οι αποχρώσεις που παίρνει αυτή η μάζα σχετίζονται με τη σειρά με την οποία ο χρήστης ρίχνει τα χρώματα. Κάθε φορά που επιτυχγάνεται ένας σχεδιασμός το επίπεδο δυσκολίας αυξάνεται.

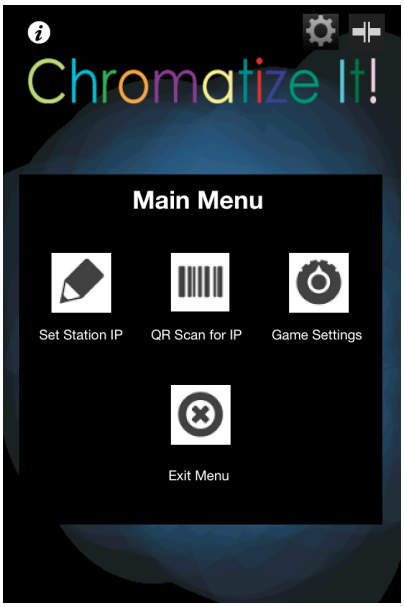


Εικόνα 6.7 Το “ταμπλώ” του Chromatize It! σε έναν Web Browser

Στην εικόνα παραπάνω φαίνεται το ταμπλώ του Chromatize It. Δεξιά φαίνεται η μάζα που πρέπει να χρωματιστεί, και αριστερά φαίνονται τα διαθέσιμα χρώματα, το χρώμα-στόχος, και τα χρώματα που πρέπει ένας χρήστης να ρίξει για να κερδίσει.

Στη συσκευή του παίκτη φαίνονται τα εξής:

	Το βασικό interface του παιχνιδιού. Το φόντο δηλώνει το χρώμα που έχει ο χρήστης στο “πινέλο” του. Ο παίκτης έχει το κίτρινο χρώμα.
	Ο παίκτης έχει το μπλε χρώμα.

	Το βασικό menu επιλογών του χρήστη.

Πίνακας 6.7 Εικόνες της iOS εφαρμογής του Chromatize It!

Και σε αυτό το παιχνίδι το φόντο της εφαρμογής χρησιμοποιείται για να δηλώσει κάτι, στην προκειμένη περίπτωση το χρώμα που έχει ο χρήστης στη διάθεση του.

7 Μελλοντικές Κατευθύνσεις

Μια προφανής μελλοντική κατεύθυνση για την πλατρφόρμα είναι η υλοποίηση κι άλλων παιχνιδιών εκτός από τα 2 παιχνίδια που παρουσιάστηκαν.

Μια άλλη σημαντική προσθήκη θα ήταν η υποστήριξη περισσότερων συσκευών, είτε αυτά είναι smartphones (Symbian, Windows 8) είτε αυτά είναι συσκευές δικτύων αισθητήρων, όπως για παράδειγμα το Arduino.

Φυσικά μια σημαντική κατεύθυνση θα μπορούσε να είναι μια υποδομή στην πλατφόρμα η οποία να επιτρέπει συλλογή στατιστικών δεδομένων από τα παιχνίδια που παίχτηκαν για ανάλυση και αξιολόγηση αυτής. Επίσης θα μπορούσε να εισαχθεί περισσότερο η έννοια του social networking στην πλατρφόρμα. Ίσως θα ήταν καλό αν οι χρήστες είχαν τη δυνατότητα να έχουν δικό τους προφίλ και δικό τους κοινωνικό δίκτυο στα πλαίσια της πλατρφόρμας.

ΒΙΒΙΛΟΓΡΑΦΙΑ

[1] HTML5: http://www.w3.org/TR/2011/WD-html5-20110525/

[2] Google Web Toolkit: http://www.gwtproject.org/

[3] AJAX: http://en.wikipedia.org/wiki/Ajax_(programming)

[4] Google Protocol Buffers: https://developers.google.com/protocol-buffers/

[5] Processing.js: http://processingjs.org/

[6] Processing: http://processing.org/

[7] WebSocket Protocol: http://tools.ietf.org/html/rfc6455

[8] SunSPOT: http://www.sunspotworld.com/

[9] Squawk Virtual Machine: http://en.wikipedia.org/wiki/Squawk_virtual_machine

[10] Ioannis Chatzigiannakis, Georgios Mylonas, Panagiotis Kokkinos, Orestis Akribopoulos, Marios Logaras, Irene Mavrommati, “Implementing multiplayer pervasive installations based on mobile sensing devices: Field experience and user evaluation from a public showcase“, 2011

[11] Ορέστης Ακριβόπουλος, “Μελετη και Αναπτυξη ενος Διαχυτου Συστηματος Ψυχαγωγιας με χρηση Ασυρματων Δικτυων αισθητήρων”, 2009

[12] Resin Web Server: http://www.caucho.com/resin-web-server/

[13] Μάριος Λογαράς, “Μελέτη και Υλοποίηση Πλατφόρμας Ανάπτυξης Παιχνιδιών Περιρρέουσας Υπολογιστικής Νοημοσύνης με Χρήση Ασύρματων Δικτύων Αισθητήρων”, 2010