Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Εφαρμοσμένης Πληροφορικής & Πολυμέσων

Πτυχιακή Εργασία

Μοντελοκεντρική Ανάπτυξη Σύγχρονων Συνεργατικών Παιγνίων για Πολλαπλά Περιβάλλοντα

"η περίπτωση χρήσης της ναυμαχίας"

Τσούτσας Γιώργος - Α.Μ. : 2542
Βίγκος Ιωάννης - Α.Μ. : 2389

Εισηγητής:

Βιδάκης Νικόλας

Ηράκλειο - εαρινό εξάμηνο 2015

Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Εφαρμοσμένης Πληροφορικής & Πολυμέσων

Πτυχιακή Εργασία

Μοντελοκεντρική Ανάπτυξη Σύγχρονων Συνεργατικών Παιγνίων για Πολλαπλά Περιβάλλοντα

"η περίπτωση χρήσης της ναυμαχίας"

Τσούτσας Γιώργος (Α.Μ. : 2542) & Βίγκος Ιωάννης (Α.Μ. : 2389)

Επιβλέπων Καθηγητής: **Βιδάκης Νικόλας**

Επιτροπή Αξιολόγησης:

....

Ηράκλειο - εαρινό εξάμηνο 2015

i. Abstract

Model-Based Development of Modern Cooperative Games for Multiple Environments: "The Battleship Game Scenario"

The purpose of this thesis is to familiarize the student with advanced software development methods, able to adapt and perform under alternative user interfaces. This becomes as imperative while given the surfeit of available heterogeneous computing devices, which users daily use to perform increasingly greater number of tasks. The development of gaming is particularly important by demand of users, at first place, as they willing to have access under all conditions, and also companies are struggling to cover possible wider range of users.

However, conventional development methods adopted for software development (toolkit-based development) enormously fail to cover the range of emerging requirements for every supported interface, eventually forcing a separate development situation in any particular case, using different toolkits and either programming languages. This significantly increases the complexity and difficulty of developing and also makes harder to achieve consistency between alternative products of each individual case, while the cost of development is increasing and leads to separately solutions for any occasion (adhoc & non-reusable solutions). Especially when we are talking about versatile gaming, situation is further complicated since the interfaces, that have involved users (participants), must be kept continuously synchronized, despite the potential heterogeneities (assembled by different interactive objects) and possible inconsistencies among them as a result of adaptation on the conditions of use (different screen sizes, platforms, blind, non-expert, kid, etc..). The model-based development on the other hand offers a better software development framework that deals with heterogeneous usage environments. However, to achieve this despite the latest progress, model-based development is limited to generalizations that leads to limited cover of more complex interfaces, especially gaming interfaces and modern collaborative where concepts management meetings, copies of articles, awareness, etc.. must be taken in mind additionally.

The aim of this thesis is to familiarize the student with the challenges and demands of modern development and omnipresent software. More specifically, this thesis aims to take advantage of model-based offered benefits in software development where specific extensions will be implemented in order to enable the interface assembly composed of the most sophisticated form-based interactive objects and later be able to achieve synchronization on abstract level so that potentially inconsistent interfaces remain synchronized. As a proof of concept scenario we select the classic game of Battleship in which potential users, under alternative user interfaces (java / desktop, android / mobile), will be able to interact and remain synchronized despite inconsistencies of their final interfaces, as a result of their adjustment in any user interface (context of use).

i.i. TLA's you need:

- ***TLA:*** Three letter acronym.
- ***UML:*** The Unified Modeling Language is an industry standard visual language for modeling software systems. These models capture knowledge about a system at various abstraction levels, ranging from requirements and analysis models

to design models. This means that modelers can specify software systems using higher-level domain-oriented concepts that abstract away much of the underlying implementation technology used to realize such systems. UML enables automated tools that interchange and transform models as part of the process and generate a system's imple-mentation artifacts (typically source code and metadata).

- ***MDA:*** The Model-Driven Architecture is a set of OMG standards that en-ables the specification of models and their transformation into other models and complete systems. MDA separates subject matters so that ap-plication-oriented models are independently reusable across multiple im-plementations and vice versa.

- ***OMG:*** The Object Management Group is an international, not-for-profit industrial consortium that creates and maintains software interoperability specifications. ***[12]*** The OMG's specifications include the UML modeling nota-tion, XMI (XML metadata interchange), CORBA (common object request broker architecture) middleware, nd dozens of domain-specific interop-erability specifications in such areas as transortation, life sciences, telecommunications, and manufacturing.

- ***PIM:*** A platform-independent model is a model that contains no reference to the platforms on which it depends. (MDA classifies the relative rela-tionship between two models in terms of the platform, the set of technolo-gies a model assumes to exist.)

- ***IDE:*** An integrated development environment (IDE) or interactive development environment is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger.

ii. Περιγραφή

Μοντελοκεντρική Ανάπτυξη Σύγχρονων Συνεργατικών Παιγνίων για Πολλαπλά Περιβάλλοντα: "Η Περίπτωση Χρήσης της Ναυμαχίας"

Σκοπός της συγκεκριμένης πτυχιακής είναι να εξοικειώσει το φοιτητή με προηγμένες μεθόδους ανάπτυξης λογισμικού ικανού να προσαρμοστεί και εκτελεστεί υπό εναλλακτικά περιβάλλοντα χρήσης. Το τελευταίο καθίσταται ως επιτακτική ανάγκη δεδομένης της υπερπληθώρας διαθέσιμων ετερογενών υπολογιστικών συσκευών τις οποίες οι χρήστες σε καθημερινή βάση χρησιμοποιούν για να εκτελέσουν ολοένα και μεγαλύτερο πλήθος καθηκόντων. Η ανάπτυξη παιγνίων αποτελεί ιδιαιτέρα σημαντική συνιστώσα ως προς αυτή την κατεύθυνση δεδομένης της ζήτησης αφενός των χρηστών να τα έχουν άμεσα διαθέσιμα υπό οποιεσδήποτε συνθήκες χρήσης και αφετέρου των εταιριών οι οποίες αγωνίζονται να καλύψουν όσο το δυνατόν μεγαλύτερη γκάμα χρηστών.

Ωστόσο οι συμβατικές μέθοδοι ανάπτυξης που υιοθετούνται για την ανάπτυξη λογισμικού (toolkit-based development) αποτυγχάνουν γηγενώς να καλύψουν το εύρος των αναδυόμενων απαιτήσεων για κάθε ένα τελικώς υποστηριζόμενο περιβάλλον χρήσης αναγκάζοντας σε ξεχωριστούς κύκλους ανάπτυξης με χρήση διαφορετικών εργαλειοθηκών και πολύ πιθανόν γλωσσών προγραμματισμού. Αυτό όπως είναι λογικό

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

αυξάνει σημαντικά την πολυπλοκότητα και δυσκολία της ανάπτυξης και επίτευξης συνοχής μεταξύ των εναλλακτικών τελικών προϊόντων κάθε κύκλου ενώ παράλληλα αυξάνει το κόστος ανάπτυξης και οδηγεί σε κατά περίπτωση (adhoc & non-reusable) λύσεις. Στην περίπτωση δε που μιλάμε για πολυχρηστικά παίγνια η κατάσταση περιπλέκεται ακόμη περισσότερο μιας και πρέπει οι διεπαφές των εμπλεκόμενων χρηστών (participants) να διατηρούνται συνεχώς συγχρονισμένες παρά τις δυνητικές ετερογένειες (συναρμολογημένες από διαφορετικά διαδραστικά αντικείμενα) και πιθανές ασυνέπειες που παρουσιάζουν μεταξύ τους ως αποτέλεσμα της προσαρμογής τους στις εκάστοτε συνθήκες χρήσης (διαφορετικά μεγέθη οθονών, πλατφόρμας, τυφλός, non-expert, kid, κοκ.).

Η μοντελοκεντρική ανάπτυξη από την άλλη πλευρά προσφέρει ένα γηγενώς καλύτερο πλαίσιο ανάπτυξης λογισμικού για την αντιμετώπιση ανομοιογενών περιβαλλόντων χρήσης. Ωστόσο για να το πετύχει αυτό και παρά τις τελευταίες εξελίξεις περιορίζεται σε γενικεύσεις που οδηγούν στην κάλυψη όχι πιο σύνθετων διεπαφών από form-based, πόσο μάλλον διεπαφών παιγνίων και δει σύγχρονων συνεργατικών όπου έννοιες όπως αυτές της διαχείρισης συνόδων, αντιγράφων αντικειμένων, ενημερότητας, κοκ. πρέπει να ληφθούν επιπλέον υπόψιν.

Στόχος της συγκεκριμένης πτυχιακής είναι να εξοικειώσει το φοιτητή με τις προκλήσεις και απαιτήσεις της ανάπτυξη σύγχρονου και παράλληλα πανταχού παρόντος λογισμικού. Πιο συγκεκριμένα στοχεύει στο να εκμεταλλευτεί τα προσφερόμενα της μοντελοκεντρικής ανάπτυξης λογισμικού πλεονεκτήματα στα πλαίσια της οποίας θα εφαρμοστούν συγκεκριμένες επεκτάσεις ούτως ώστε αφενός να γίνει δυνατή η συναρμολόγηση διεπαφών απαρτιζόμενων από πιο εξεζητημένα των form-based διαδραστικών αντικειμένων και αφετέρου να δύναται η επίτευξη συγχρονισμού σε αφηρημένο επίπεδο τέτοιο ούτως ώστε δυνητικώς ασυνεπείς μεταξύ τους διεπαφές να παραμένουν συγχρονισμένες.

Ως σενάριο επίδειξης της ορθότητας της προσέγγισης (proof-of-concept scenario) επιλέγεται το κλασικό παίγνιο της ναυμαχίας στα πλαίσια της οποίας δυνητικοί χρήστες αυτής υπό εναλλακτικά περιβάλλοντα χρήσης (java/desktop, android/mobile), θα δύνανται να αλληλεπιδρούν και να παραμένουν συγχρονισμένοι παρά τις όποιες ασυνέπειες των τελικών διεπαφών τους ως αποτέλεσμα της προσαρμογής αυτών σε κάθε περιβάλλον χρήσης (context of use).

ii.i. Επεξηγήσεις ακρώνυμων:

- ***TLA***: Ακρώνυμο τριών λέξεων.

- ***UML:*** Η Unified Modeling Language είναι μια βιομηχανική πρότυπη οπτική γλώσσα που προσομοιώνει συστήματα λογισμικού. Τα μοντέλα αυτά συλλαμβάνουν γνώσεις σχετικά με ένα σύστημα σε διάφορα επίπεδα αφαίρεσης, που κυμαίνονται από τις απαιτήσεις και τα μοντέλα ανάλυσης, για να σχεδιάσουν ύστερα τα μοντέλα αυτά. Αυτό σημαίνει ότι οι μοντελιστές για να καθορίσουν τα συστήματα λογισμικού χρησιμοποιούν υψηλότερου επιπέδου γλώσσες με γνώμονα έννοιες που απομονώνουν ένα μεγάλο μέρος της υφιστάμενης εφαρμοσμένης τεχνολογίας που χρησιμοποιείται για την υλοποίηση τέτοιων συστημάτων. Η γλώσσα UML παρέχει αυτοματοποιημένα εργαλεία που εναλλάσσουν και μετατρέπουν τα μοντέλα σε μέρος της διαδικασίας ώστε να δημιουργήσουν εφαρμογές ενός συστήματος. (συνήθως πηγαίο κώδικα και μετα-δεδομένα).

- ***MDA:*** Η μοντελοκεντρική αρχιτεκτονική είναι ένα σύνολο προτύπων του

οργανισμού OMG που επιτρέπει διάφορες δυνατότητες στις προδιαγραφές των μοντέλων, και τη μετατροπή τους σε άλλα μοντέλα καθώς και σε ολοκληρωμένα συστήματα. Η MDA διαχωρίζει τα αντικείμενα, έτσι ώστε τα μοντέλα - εφαρμογές να είναι ανεξάρτητα και επαναχρησιμοποιήσιμα σε πολλαπλές εφαρμογές και αντίστροφα.

- ***OMG:*** Η OMG είναι μια διεθνής μη κερδοσκοπική βιομηχανική κοινοπραξία, ***[12]*** που δημιουργεί και διατηρεί τις προδιαγραφές διαλειτουργικότητας του λογισμικού. Οι προδιαγραφές της OMG περιλαμβάνουν τη σημειογραφική μοντελοποίηση UML, XMI (XML ανταλλαγή μετα-δεδομένων), CORBA (common object request broker architecture middleware), και δεκάδες συγκεκριμένους τομείς προδιαγραφών διαλειτουργικότητας σε τομείς όπως επιστήμες της ζωής, τις τηλεπικοινωνίες, και τις κατασκευές.

- ***PIM:*** Το μοντέλο που είναι ανεξάρτητο από την πλατφόρμα. Δηλαδή ένα μοντέλο που δεν περιλαμβάνει καμία αναφορά για τις πλατφόρμες από τις οποίες εξαρτάται. (Η MDA κατατάσσει την συγγενική σχέση μεταξύ δύο μοντέλων σε σχέση με την πλατφόρμα, το σύνολο των τεχνολογιών όπου ένα μοντέλο υποθέτουμε πως υπάρχει.)

- ***IDE:*** Ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) ή διαδραστικό περιβάλλον ανάπτυξης είναι μια εφαρμογή λογισμικού που παρέχει ολοκληρωμένες εγκαταστάσεις για προγραμματιστές υπολογιστών για την ανάπτυξη λογισμικού. Ένα IDE αποτελείται συνήθως από ένα πρόγραμμα επεξεργασίας πηγαίου κώδικα, αυτοματοποιημένα εργαλεία ανάπτυξης και ένα πρόγραμμα εντοπισμού σφαλμάτων.

iii. Keywords

MDA, model-driven development, model-driven architecture, UML diagram, battleship development, Java Swing, TCP java sockets, cooperative games, MVC development

iv. Λέξεις Κλειδιά

μοντελοκεντρική ανάπτυξη, μοντελοκεντρική αρχιτεκτονική, διάγραμμα UML, παιχνίδι ναυμαχίας, συνεργατικά παιχνίδια

v. Πίνακας Περιεχομένων

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

vi. Ευρετήριο Εικόνων

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

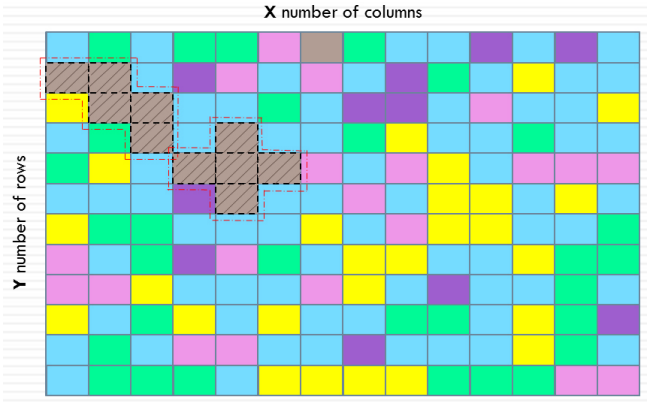
Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

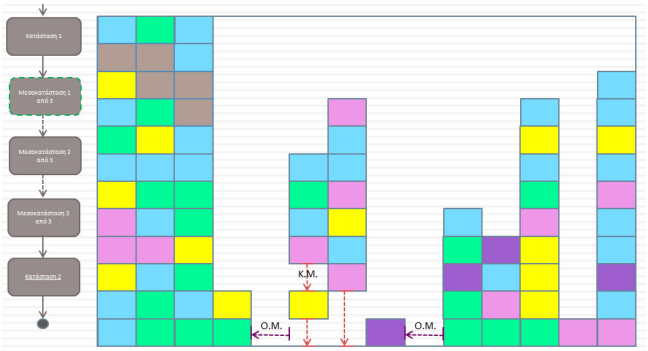
1. Εισαγωγή

1.1. Κίνητρο επιλογής του συγκεκριμένου παιγνίου:

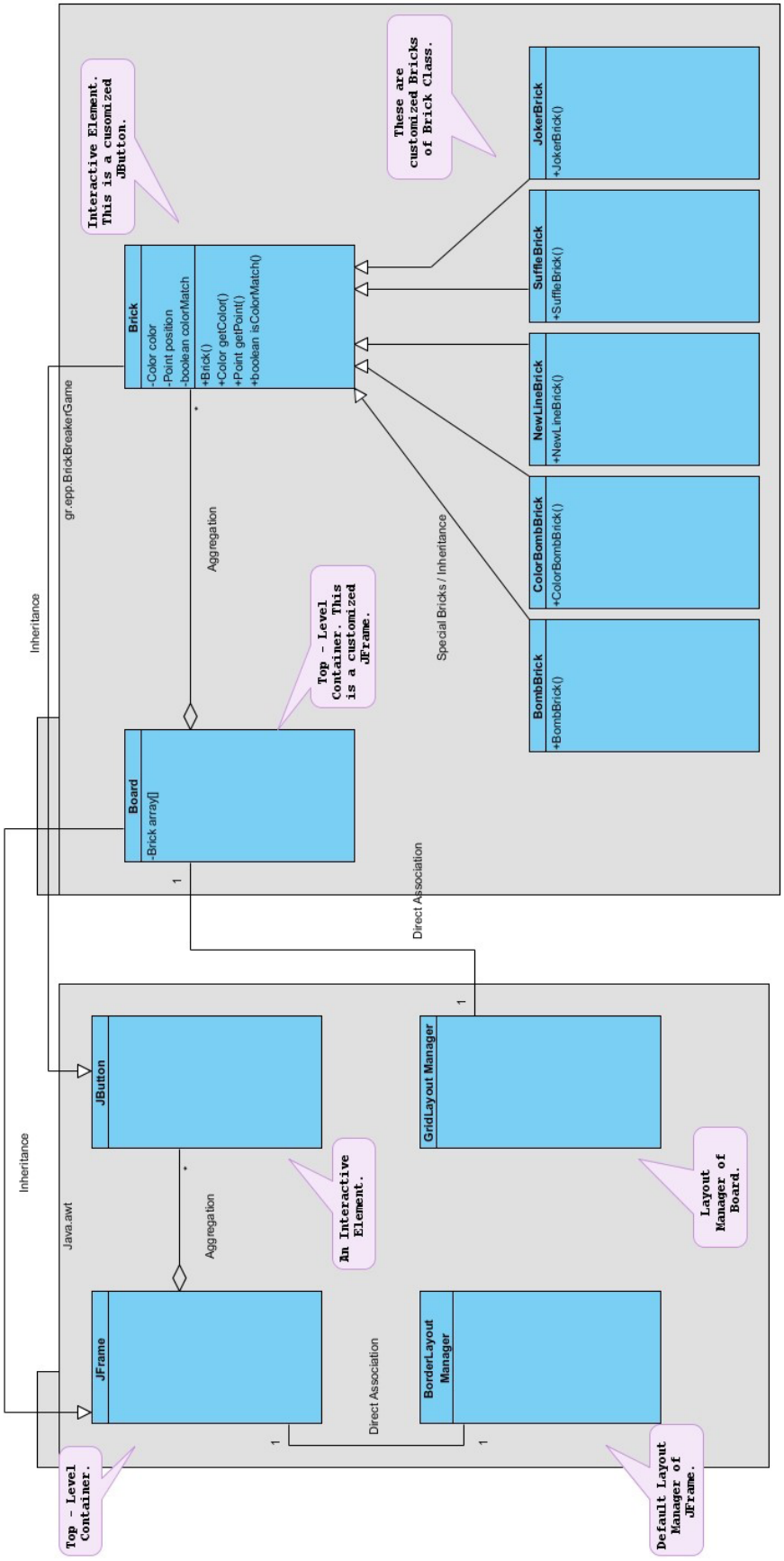
Επιλέξαμε να υλοποιήσουμε παιχνίδι διότι βρήκαμε μεγάλο ενδιαφέρον σε ένα παλαιότερο project που υλοποιήθηκε το Εαρινό Εξάμηνο του 2012, στο εργαστήριο Αντικειμενοστραφής Προγραμματισμός Ι. Το παιχνίδι που υλοποιήσαμε τότε, με τίτλο "**The Brick Breaker Game**" ήταν ότι χρειαζόμασταν από καιρό, καθ' όλη την διάρκεια των σπουδών μας στο ίδρυμα, για να μας κεντρίσει το ενδιαφέρον σχετικά με τη Java. Πάντα με τους καθηγητές μας αρωγούς, ανακαλύψαμε πως κάτι μας είχε κερδίσει τελικά. Θα έλεγε κανείς ότι γεννήθηκε μια "μανία" με το να μαθαίνουμε συνεχώς περισσότερα για τον Αντικειμενοστραφή Προγραμματισμό και τις δυνατότητες του. Πάντα φάνταζε ως ένα εργαλείο με ανεξάντλητες δυνατότητες και προσπαθούσαμε να δημιουργούμε από το μηδέν. Δεν ήταν μάλιστα λίγες οι φορές που χρησιμοποιήσαμε τις γνώσεις μας για να δημιουργήσουμε εργαλεία που θα μας βοηθούσαν σε άλλα εργαστήρια, όπως π.χ. έναν mini υπολογιστή για το εργαστήριο Κινητές & Προσωπικές Τηλεπικοινωνίες σε επόμενο εκπαιδευτικό εξάμηνο.

Θα θέλαμε να σταθούμε σε ένα σημαντικό σημείο "κλειδί". Θαρρούμε πως μας βοήθησε πολλές φορές να συλλάβουμε και να καταλάβουμε πλήρως του τι πρόκειται να υλοποιήσουμε στη συνέχεια, κάθε φορά, η τεχνική της γλώσσας UML.
Η γλώσσα UML, όπως θα δούμε ύστερα, είναι ένα βιομηχανικό πρότυπο οπτικής γλώσσας για να προσομοιώνει τα συστήματα λογισμικού που πρόκειται να υλοποιηθούν. Τα μοντέλα αυτά συλλαμβάνουν γνώσεις στο σύνολό τους σχετικά με ένα σύστημα σε διάφορα επίπεδα αφαίρεσης, που κυμαίνονται από τις απαιτήσεις και την ανάλυση των μοντέλων, ώστε να σχεδιαστούν τελικώς τα μοντέλα.





εικόνα 1 & 2 *- project: the Brick Breaker Game*



εικόνα 3 *- διάγραμμα UML του Brick Breaker Game*

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

Σε συνάρτηση λοιπόν με το ότι η Ναυμαχία είναι ένα εξαιρετικά όμορφο, απλό, αλλά και έξυπνο παιχνίδι στρατηγικής, επίσης έχοντας υπόψη ότι πολλάκις έχει υλοποιηθεί σαν εφαρμογή σε διάφορα sites στο internet, θελήσαμε να δοκιμάσουμε και 'μεις να παντρέψουμε τα σοβαρά πλεονεκτήματα της Μοντελο-Κεντρικής Ανάπτυξης λογισμικού με τις απαιτήσεις αυτού του απλού παιγνίου.

Όπως προαναφέρθηκε, είναι ένα εύκολο παιχνίδι ως προς τον τρόπο που λειτουργεί καθώς και τη λογική του. Άρα σκεφτήκαμε πως αν θέλουμε να παρουσιάσουμε ξεκάθαρα τα πλεονεκτήματα που συγκεντρώνει η τεχνική του πολυμορφισμού (**versatile**) και τον αναλυτικό τρόπο σκέψης της Μοντελο-Κεντρικής Ανάπτυξης λογισμικού (**MDA Development**) σίγουρα θα προσέφερε πρόσφορο έδαφος η επιλογή του συγκεκριμένου.

Επίσης, αρχικά, βρήκαμε τους εαυτούς μας σχετικά σίγουρους και θαρραλέους ως προς την υλοποίηση του και σε Android/mobile περιβάλλον χρήσης, περιβάλλον στο οποίο είναι η παρθενική μας προσέγγιση. Σκεφτήκαμε πως αποτελεί ένα καλό παράδειγμα το συγκεκριμένο παιχνίδι ώστε να πιέσουμε τους εαυτούς μας να βρούμε λύσεις, να σβήσουμε, να γράψουμε, να εκνευριστούμε, να ξανασχεδιάσουμε και τέλος να μας μείνουν αρκετές εμπειρίες από την υλοποίηση του παιχνιδιού και σε περιβάλλον Android.

1.2. Σενάριο αναφοράς: Το παιχνίδι της Ναυμαχίας

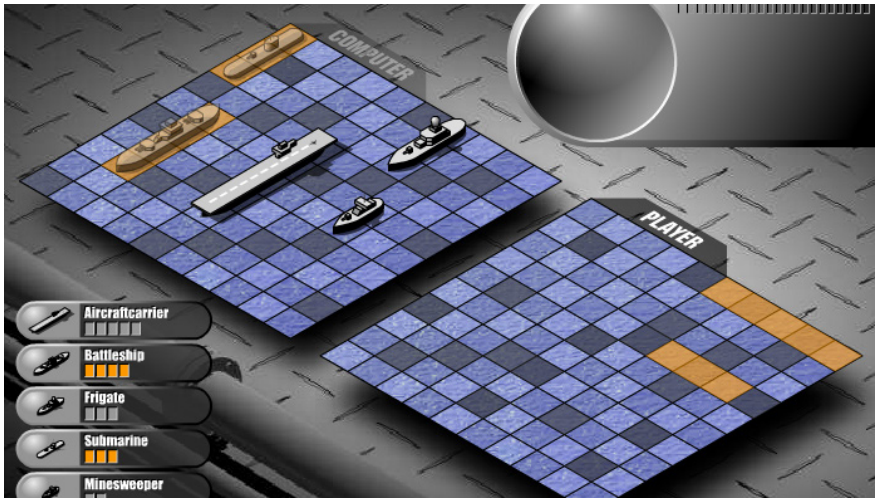
Η Ναυμαχία είναι ένα παιχνίδι εικασίας για δύο παίκτες. ***[1]*** & ***[2]*** Είναι γνωστό σε όλο τον κόσμο ως ένα παιχνίδι για μολύβι και χαρτί που χρονολογείται από τον Πρώτο Παγκόσμιο Πόλεμο. Στη δεκαετία του 1930 επίσης, κυκλοφόρησε ως ένα επιτραπέζιο παιχνίδι. Το παιχνίδι της Ναυμαχίας πιστεύεται ότι έχει τις ρίζες του στη Γαλλία, κατά τη διάρκεια του Ά Παγκοσμίου Πολέμου, αλλά επίσης αναφορές υπάρχουν πως παιζόταν και από Ρώσους Αξιωματικούς πολύ πριν τον πόλεμο.

εικόνα 4 *- υπολογιστής Z80 Compucolor (1979)*
source: *http://oldcomputers.net/compucolor-ii.html*

Η Ναυμαχία ήταν ένα από τα πρώτα παιχνίδια που θα παρουσιαζόταν ως ένα παιχνίδι στον υπολογιστή, με μια έκδοση που κυκλοφόρησε για το Z80 Compucolor το 1979, και του Battle Zone Atari κατόπιν, στις αρχές της δεκαετίας του 1980.

εικόνα 5 *- Battle Zone Atari*
source: *https://atariage.com*

Σε πάρα πολλές εκδόσεις για υπολογιστή έχει παραχθεί από τότε. Εμφανίσεις του παιχνιδιού, ως εφαρμογές, έχουμε σε πολλές υπηρεσίες κοινωνικής δικτύωσης. Η Ναυμαχία ήταν επίσης προϊόν της Hasbro Family Game Night, για το PlayStation 2 και Wii, καθώς και για το Xbox 360 (Xbox Live Arcade).



***εικόνα 6 & 7** - παιχνίδι Ναυμαχίας στο web*
***source:** http://ru.battleship-game.org/*

1.3. Σκοπός του παιχνιδιού:

Αντικειμενικός Σκοπός είναι να αναπτύξει ο παίκτης τα πλοία του με στρατηγικό τρόπο στο πεδίο, και να προσπαθήσει να βυθίσει τα αντίστοιχα πλοία του αντιπάλου.

Στη διάθεσή του έχει: ένα **αεροπλανοφόρο** (5 blocks), ένα **θωρηκτό** (4 blocks), ένα **υποβρύχιο** (3 blocks), μια **φρεγάτα** (2 blocks) και μια **τορπιλάκατο** (1 block).

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

1.4. Δομή τόμου:

Η συνέχεια της πτυχιακής έχει οργανωθεί στα ακόλουθα κεφάλαια:

- Στο **Κεφάλαιο 2** παρουσιάζονται αναλυτικά οι τεχνολογίες που χρησιμοποιήθηκαν για να αναπτυχθεί το project μας από το μηδέν. Αναφέρονται ιστορικά στοιχεία για την java, καθώς και τα εργαλεία της συνοπτικά, αλλά και χρήσιμες πληροφορίες γι' αυτήν. Ύστερα ακολουθεί μια αναλυτική αναφορά για τον IDE μας, το πρόγραμμα NetBeans, και τέλος αναλυτικές πληροφορίες επίσης για την γλώσσα μοντελοποίησης UML.
- Στο **Κεφάλαιο 3** παρουσιάζεται η μεθοδολογία ανάπτυξης που ακολουθήσαμε. Δηλαδή η φιλοσοφία και η στρατηγική μας επάνω στον τρόπο που επιλέξαμε να ολοκληρώσουμε την πτυχιακή μας εργασία. Αναφέρονται όροι, προσόντα και μειονεκτήματα της μοντελοκεντρικής ανάπτυξης λογισμικού, καθώς και της μοντελοκεντρικής αρχιτεκτονικής γενικότερα. Τέλος αρκετές πληροφορίες και εικόνες για το Java Swing και τα TCP Java Sockets γενικότερα.
- Στο **Κεφάλαιο 4** παρουσιάζεται εκτενώς η περίπτωση χρήσης της Ναυμαχίας ως το πρότυπο που επιλέξαμε εμείς προκειμένου να αναδείξουμε τα πλεονεκτήματα της μοντελοκεντρικής ανάπτυξης λογισμικού και του πολυμορφισμού που μπορεί να εφαρμοστεί σε έναν κώδικα, ούτως ώστε να καλύπτει γηγενώς διάφορους τύπους χρηστών. Αφού γίνει μια σχετική εισαγωγή, ύστερα εξηγούνται αναλυτικά οι λειτουργίες του κλάσεων του project.
- Στο **Κεφάλαιο 5** παρουσιάζονται ιδέες και προτάσεις δικές μας που πιστεύουμε μπορούν να δώσουν ακόμα πιο ενδιαφέρουσα πορεία στο project αν υλοποιηθούν. Η τεχνική MVC, η ολοκλήρωση του παιχνιδιού σε περιβάλλον android είναι μερικά από αυτά.

2. Τεχνολογίες που χρησιμοποιήθηκαν

2.1. Η γλώσσα προγραμματισμού Java

2.1.1. Ιστορικά και εισαγωγικά:

Η γλώσσα της Java αρχικά αναπτύχθηκε από την Sun Microsystems υπό την αιγίδα των James Gosling και Bill Joy, σαν μέρος ενός ερευνητικού έργου ανάπτυξης λογισμικού για ηλεκτρονικές συσκευές καταναλωτικού επιπέδου (π.χ video, τηλεόραση). ***[3]*** Ο τρόπος αυτός χρησιμοποίησης της, την μετέτρεψε σε μία ιδανική γλώσσα για την διανομή εκτελέσιμων προγραμμάτων μέσω του WWW (Παγκόσμιου Ιστού) καθώς επίσης σε μία γενικού σκοπού γλώσσα προγραμματισμού για την ανάπτυξη προγραμμάτων τα οποία θα είναι εύκολα στην χρήση και θα μπορούν να μεταφέρονται σε διαφορετικά λειτουργικά συστήματα. Αν και χρησιμοποιήθηκε από την Sun σε πολλές εφαρμογές (με το όνομα Oak) με σκοπό την δημιουργία προϊόντων για την ηλεκτρονική αγορά, ενδιαφέρον προκάλεσε στο κοινό μόνο όταν συνδυάστηκε με το πρόγραμμα ανάγνωσης ιστοσελίδων (browser) της HotJava. Αυτό είχε ως

αποτέλεσμα, η γλώσσα αυτή να συνδεθεί στενά με την ανάπτυξη μικρό - εφαρμογών στο διαδίκτυο και συγκεκριμένα στον Παγκόσμιο Ιστό (WWW). Η πραγματική όμως απογείωση της δημοτικότητας της Java ξεκίνησε όταν η Netscape ενσωμάτωσε την δυνατότητα της HotJava να τρέχει μικρό - εφαρμογές μέσα στο δικό της πρόγραμμα ανάγνωσης ιστοσελίδων (browser). Το γεγονός, ότι τα τελευταία χρόνια η Java χρησιμοποιείται κυρίως για την ανάπτυξη μικρό - εφαρμογών δεν σημαίνει ότι η γλώσσα αυτή δεν είναι κατάλληλη για την δημιουργία ολοκληρωμένων εφαρμογών. Εξάλλου αυτό αποδεικνύεται από το γεγονός ότι τα περισσότερα εργαλεία της έχουν γραφτεί με την Java. Μάλιστα, σύμφωνα με την θεωρεία ανάπτυξης μεταγλωττιστών, μία γλώσσα έχει "**ενηλικιωθεί**" όταν ο μεταγλωττιστής της μπορεί να γραφτεί από την ίδια την γλώσσα. Συνεπώς η Java ως γλώσσα προγραμματισμού έχει ενηλικιωθεί. Αυτό που θα πρέπει κανείς να θυμάται για την Java είναι ότι πρόκειται για μία καλά οργανωμένη και καλά εκφρασμένη γλώσσα προγραμματισμού που έχει δανειστεί πολλά θετικά χαρακτηριστικά από άλλες γλώσσες όπως τη C++, τη SmallTalk και τη Lisp, αφαίρεσε όμως όλα εκείνα τα στοιχεία αυτών των γλωσσών που ίσως οδηγούσαν σε σύγχυση τους χρήστες. Η Java σχεδιάστηκε με σκοπό την ανάπτυξη εφαρμογών που θα τρέχουν σε ετερογενή δικτυακά περιβάλλοντα.

Η Java έχει τα ακόλουθα χαρακτηριστικά:

- Αντικειμενοστραφής (ομοιότητες εντολών με τη C++).
- Δημιουργία ανεξάρτητων εφαρμογών και applets (applet = προγράμματα που περιλαμβάνονται σε HTML σελίδες και εκτελούνται από τον Web Browser).
- Είναι Interpreted γλώσσα. Αυτό σημαίνει ότι ο java compiler δεν παράγει εκτελέσιμο κώδικα αλλά μια μορφή ψευδοκώδικα (bytecode) το οποίο από μόνο του δεν τρέχει σε καμία μηχανή. Προκειμένου λοιπόν να εκτελεστεί απαιτείται η χρήση ενός interpreter (=διερμηνέα) για να μετατρέψει το bytecode σε πραγματικό εκτελέσιμο κώδικα. Αυτό το χαρακτηριστικό δίνει τη δυνατότητα στα java bytecodes να μπορούν να τρέξουν σε οποιοδήποτε μηχάνημα, κάτω από οποιοδήποτε λειτουργικό, αρκεί να έχει εγκατασταθεί ένας java interpreter. Επίσης ένα άλλο χαρακτηριστικό του java bytecode είναι το μικρό του μέγεθος, (μόλις λίγα Kilobytes). Αυτό το κάνει ιδανικό για μετάδοση μέσω του δικτύου.
- Κατανεμημένη (distributed). Δηλαδή ένα πρόγραμμα σε Java είναι δυνατό να το φέρουμε από το δίκτυο και να το τρέξουμε. Επίσης είναι δυνατό διαφορετικά κομμάτια του προγράμματος να έρθουν από διαφορετικά sites.
- Ασφαλής (secure). Στο δίκτυο όμως ελλοχεύουν πολλοί κίνδυνοι για τον χρήστη - παραλήπτη μιας δικτυακής εφαρμογής, γι' αυτό η Java έχει σχεδιαστεί έτσι ώστε να ελαχιστοποιείται η πιθανότητα προσβολής του συστήματος του χρήστη από κάποιο applet γραμμένο για τέτοιο σκοπό.
- Είναι multithreaded. Η Java υποστηρίζει εγγενώς την χρήση πολλών threads. Προκειμένου να το επιτύχει αυτό σε συστήματα με έναν επεξεργαστή, το Java runtime system (interpreter) υλοποιεί ένα δικό χρονοδρομολογητή (scheduler), ενώ σε συστήματα που υποστηρίζουν πολυεπεξεργασία η δημιουργία των threads ανατίθεται στο λειτουργικό σύστημα. Φυσικά όλα αυτά είναι αόρατα τόσο στον προγραμματιστή όσο και στον χρήστη.
- Υποστηρίζει multimedia εφαρμογές. Με αυτό εννοούμε ότι η Java παρέχει ευκολίες στη δημιουργία multimedia εφαρμογών. Αυτό επιτυγχάνεται τόσο με την ευελιξίας της σαν γλώσσα όσο και με τις πλούσιες και συνεχώς εμπλουτιζόμενες βιβλιοθήκες της.

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

2.1.2. Τα εργαλεία της Java:

Ακολούθως παρουσιάζονται εν συντομία όλα τα εργαλεία: ***[4]***

- **javac** Είναι ο compiler της Java. Η χρήση του στο command-line είναι: javac <όνομα αρχείου>. Εδώ να σημειώσουμε ότι το javac δεν παράγει ένα αρχείο με όλον τον κώδικα, αλλά χωριστό αρχείο για κάθε κλάση. Τα αρχεία των κλάσεων ονομάζονται : <όνομα κλάσης>.class.
- **java** Είναι ο interpreter της Java. Η χρήση του είναι η εξής : java <κλάση>, πχ java myClass και όχι java myClass.class.
- **javaw** (MONO στα Windows 95/NT) Είναι παρόμοιο με το java με μόνη την διαφορά ότι δεν χρειάζεται shell για να τρέξει.
- **jdb** Είναι ο Java debuger.
- **javah** Κατασκευάζει C files και stub files για κάποια κλάση. Αυτά τα αρχεία είναι απαραίτητα όταν θέλουμε να υλοποιήσουμε κάποιες από τις μεθόδους της κλάσης σε C, πράγμα πολύ σπάνιο.
- **javap** Είναι ο Java disassembler.
- **javadoc** Είναι ένα πρόγραμμα για αυτόματη κατασκευή documentation. Είναι αρκετά χρήσιμο στην κατασκευή βοηθημάτων και τεχνικών αναφορών για εφαρμογές οποιουδήποτε μεγέθους.
- **appletviewer** Είναι ένα πρόγραμμα το οποίο μας επιτρέπει να τρέχουμε και να χρησιμοποιούμε τα διάφορα applets σε Java. Οι stand-alone εφαρμογές, ωστόσο, δεν τρέχουν στον appletviewer αλλά κατευθείαν στον java ή javaw.

2.1.3. Χρήσιμες διευθύνσεις για την Java:

Κλείνοντας το κεφάλαιο αυτό, που σκοπό είχε μόνο μια απλή αναφορά και παράθεση ιστορικών στοιχείων, παραθέτουμε κάποιες διευθύνσεις του Internet που παρέχουν πληροφορίες και χρήσιμες οδηγίες για τη γλώσσα προγραμματισμού Java, γενικότερα.

Java Tutorial:

http://docs.oracle.com/javase/tutorial/

Περιέχει αναλυτικό εκπαιδευτικό υλικό και ενσωματωμένα παραδείγματα για όλα τα θέματα της γλώσσας. Μπορείτε να αναφέρεστε σε αυτό όποτε χρειάζεται να αντλήσετε περισσότερες πληροφορίες σχετικά με κάποιο θέμα που αναφέρουν οι σημειώσεις.

Java API:

http://docs.oracle.com/javase/1.5.0/docs/api/index.html

Πολύ χρήσιμη διεύθυνση που είναι σκόπιμο να την έχετε ανοικτή όταν προγραμματίζεται σε Java. Περιέχει το Application Programming Interface της Java

(Java API) δηλαδή όλες της κλάσεις που διαθέτει η Java ομαδοποιημένες ανάλογα με τις λειτουργίες που προσφέρουν σε ενότητες (πακέτα), την περιγραφή της λειτουργίας κάθε κλάσης καθώς και των μεθόδους της.

Java Platform:

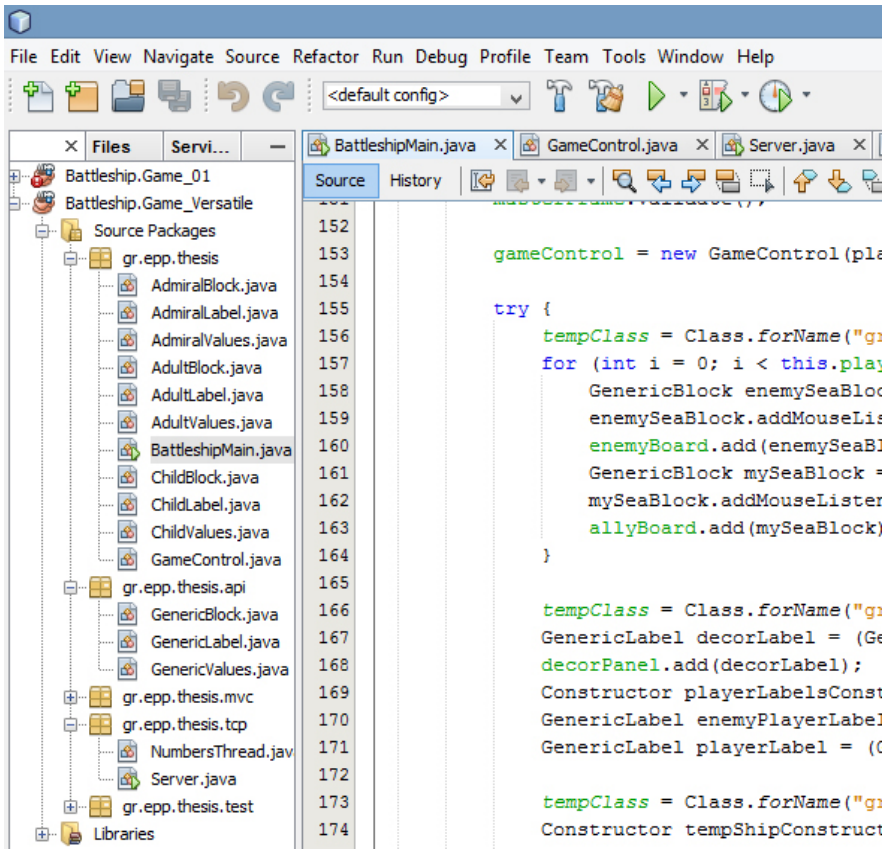
http://www.oracle.com/technetwork/java/index.html

Η Java πλατφόρμα με τα εργαλεία που αναφέρθηκαν στην παράγραφο 2.1.2. είναι το εκτελέσιμο πρόγραμμα που πρέπει να εγκαταστήσετε στον υπολογιστή σας ώστε να είστε σε θέση να προγραμματίζεται σε Java. Το συγκεκριμένο εκτελέσιμο σας εγκαθιστά και το JRE (Java Runtime Environment) στον Web Browser του υπολογιστή σας εάν αυτό δεν έχει ήδη εγκατασταθεί. Το JRE δίνει τη δυνατότητα στο Web Browser σας να εκτελεί java applets.

2.2. Εργαλείο ανάπτυξης λογισμικού NetBeans:

Το NetBeans είναι ένα επιτυχημένο ερευνητικό έργο ανοιχτής πηγής (**open source**) με μεγάλο αριθμό χρηστών, μια αναπτυσσόμενη κοινωνία, κοντά στους 100 (και πλέον!) συνεργάτες παγκοσμίως. ***[5]*** Η Sun Microsystems ίδρυσε το ερευνητικό έργο ανοιχτής πηγής NetBeans τον Ιούνιο του 2000 και συνεχίζει να είναι ο κύριος ανάδοχος.

Σήμερα δύο ερευνητικά έργα υπάρχουν: Το NetBeans IDE και το NetBeans Platform. Το NetBeans IDE είναι ένα περιβαλλοντικό ανάπτυγμα IDE - ένα εργαλείο στους προγραμματιστές για να γράψουν, να κάνουν compile, debug και να αναπτύξουν προγράμματα. Είναι γραμμένο σε Java - αλλά μπορεί να υποστηρίξει όλες τις γλώσσες προγραμματισμού. Υπάρχει επίσης ένας μεγάλος αριθμός υπομονάδων (modules) που βοηθάνε στην επέκταση της λειτουργικότητας του NetBeans IDE. Το NetBeans IDE είναι ένα ελεύθερο προϊόν δίχως περιορισμούς στον τρόπο χρησιμοποίησής του. Διαθέσιμο επίσης είναι το NetBeans Platform ένα εκτατό θεμέλιο αποτελούμενο από υπομονάδες (modular) που χρησιμοποιείται σαν βάση λογισμικού για τη



εικόνα 8 *- περιβάλλον NetBeans*

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

δημιουργία μεγάλων επιτραπέζιων (desktop) εφαρμογών. Οι ISV συνεργάτες διαθέτουν προσθήκες, επιπρόσθετα προγράμματα (plug-ins) που εύκολα συνενώνονται στο Platform και μπορούν επίσης να χρησιμοποιηθούν για την ανάπτυξη άλλων εργαλείων και λύσεων.

Και τα δύο τα προϊόντα είναι ανοιχτής πηγής (open source) και ελεύθερα για εμπορική ή μη χρήση. Ο κώδικας πηγής (source code) είναι διαθέσιμος για επαναχρησιμοποίηση κάτω από το Common Development and Distribution License (CDDL).

Το *netbeans.org* είναι το σπίτι της NetBeans κοινότητα ανοιχτής πηγής (open source community) η οποία είναι αφοσιωμένη στο χτίσιμο ενός παγκοσμίας τάξεως IDE. Το netbeans.org επιτρέπει στους χρήστες περισσότερων από 160 χωρών παγκοσμίως να είναι σε επαφή με πηγές γνώσεων και άτομα που περιβάλλουν το NetBeans.

Μπορεί κανείς να κατεβάσει την τελευταία έκδοση του NetBeans, σε αυτή τη διεύθυνση: *https://netbeans.org/downloads/*

Με αυτόν τον IDE συντάξαμε τον απαραίτητο κώδικα για το παιχνίδι της Ναυμαχίας, σε περιβάλλον Desktop.

2.3. Unified Modelling Language:

Η Unified Modelling Language (UML) είναι μια γραφική γλώσσα μοντελοποίησης. Η πρώτη έκδοση της UML έγινε διαθέσιμη το 1997 από το **Object Management Group** (OMG). ***[6]*** Ένας από τους στόχους της UML είναι να παρέχει μια σταθερή και κοινή γλώσσα σχεδιασμού για εφαρμογές λογισμικού. Η UML σύντομα έγινε αποδεκτή και αποτελεί πλέον ένα ευρέως διαδεδομένο standard. Η UML προσφέρει ένα σύνολο από διαγράμματα τα οποία μπορούν να χρησιμοποιηθούν μέσα σε μια μεθοδολογία ανάπτυξης λογισμικού προκειμένου να γίνει ευκολότερη η κατανόηση της εφαρμογής που αναπτύσσεται.

Τα διαγράμματα που ορίζονται από την UML χωρίζονται σε τρεις κατηγορίες:

Διαγράμματα συμπεριφοράς:

- Διαγράμματα περιπτώσεων χρήσης (use case diagrams).
- Διαγράμματα καταστάσεων (state machine diagrams).
- Διαγράμματα δραστηριοτήτων (activity diagrams).
- Διαγράμματα επικοινωνίας (communication diagrams).
- Διαγράμματα ακολουθίας (sequence diagrams).

Δομικά διαγράμματα:

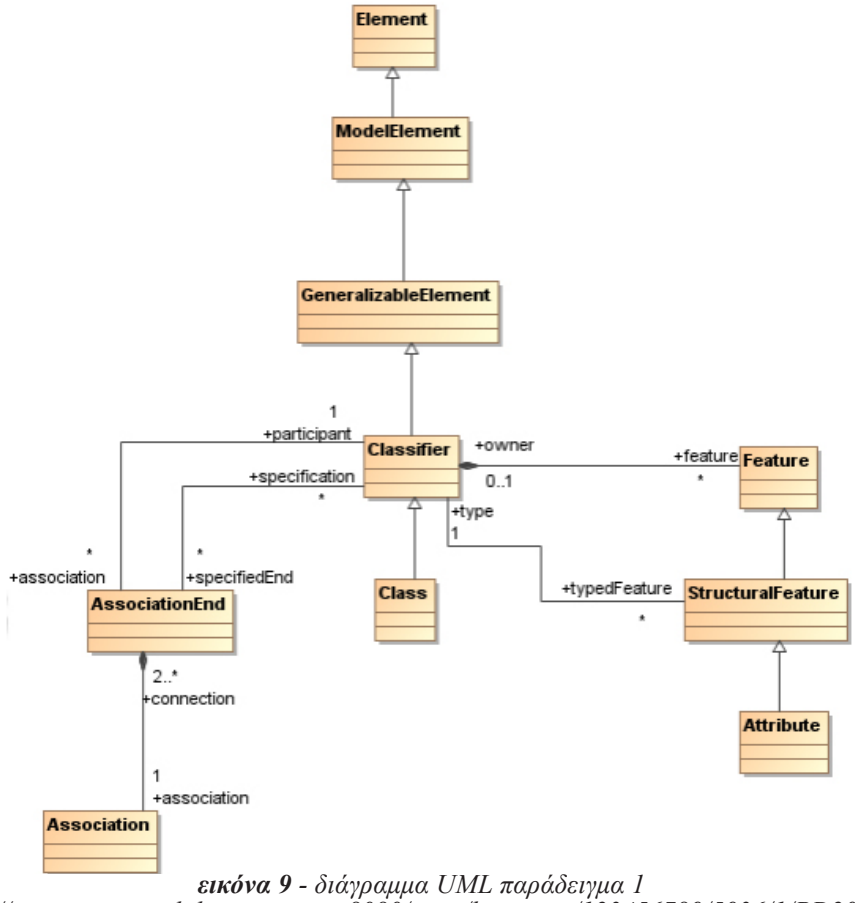
- Διαγράμματα κλάσεων (class diagrams).
- Διαγράμματα αντικειμένων (object diagrams).
- Διαγράμματα πακέτων (package diagrams).

Αρχιτεκτονικά διαγράμματα:

- Διαγράμματα εξαρτημάτων (component diagrams).
- Διαγράμματα ανάπτυξης (deployment diagrams).
- Διαγράμματα συστατικών (component diagrams).
- Διαγράμματα συνεργασίας (collaboration diagrams).

Τα διαγράμματα κλάσεων της UML είναι ο πυρήνας της αντικειμενοστραφούς ανάλυσης και σχεδίασης. Τα διαγράμματα αυτά δείχνουν τις κλάσεις του συστήματος, τις σχέσεις τους (συμπεριλαμβανόμενων της κληρονομικότητας – inheritance - , της συνάθροισης – aggregation - και των συσχετίσεων - associations), τα πεδία και τις λειτουργίες των κλάσεων. Τα διαγράμματα κλάσεων χρησιμοποιούνται για πολλούς διαφορετικούς σκοπούς όπως είναι η περιγραφή του μοντέλου ενός προβλήματος, την περιγραφή εννοιών και την λεπτομερή σχεδίαση μοντέλων. Τα διαγράμματα καταστάσεων αναπαριστούν τις διάφορες καταστάσεις που ένα αντικείμενο μπορεί να πάρει, καθώς και τις μεταβάσεις ανάμεσα σε αυτές τις καταστάσεις.

Μια κατάσταση (state) αντιπροσωπεύει ένα στάδιο στη συμπεριφορά του αντικειμένου. Όπως και στα διαγράμματα δραστηριοτήτων της UML είναι δυνατό να υπάρχουν αρχικές και τελικές καταστάσεις. Μια αρχική κατάσταση ομομάζεται και κατάσταση δημιουργίας, και είναι η κατάσταση που έχει το αντικείμενο όταν δημιουργείται, ενώ μια τελική κατάσταση είναι μια κατάστασή απ' όπου δεν υπάρχουν μεταβάσεις εξόδου. Μια μετάβαση είναι η μεταβολή της κατάστασης του αντικειμένου και πραγματοποιείται από κάποιο γεγονός είτε εσωτερικό είτε εξωτερικό ως προς το αντικείμενο.



εικόνα 9 *- διάγραμμα UML παράδειγμα 1*
source: *http://artemis-new.cslab.ece.ntua.gr:8080/jspui/bitstream/123456789/5936/1/PD2008-0014.pdf*

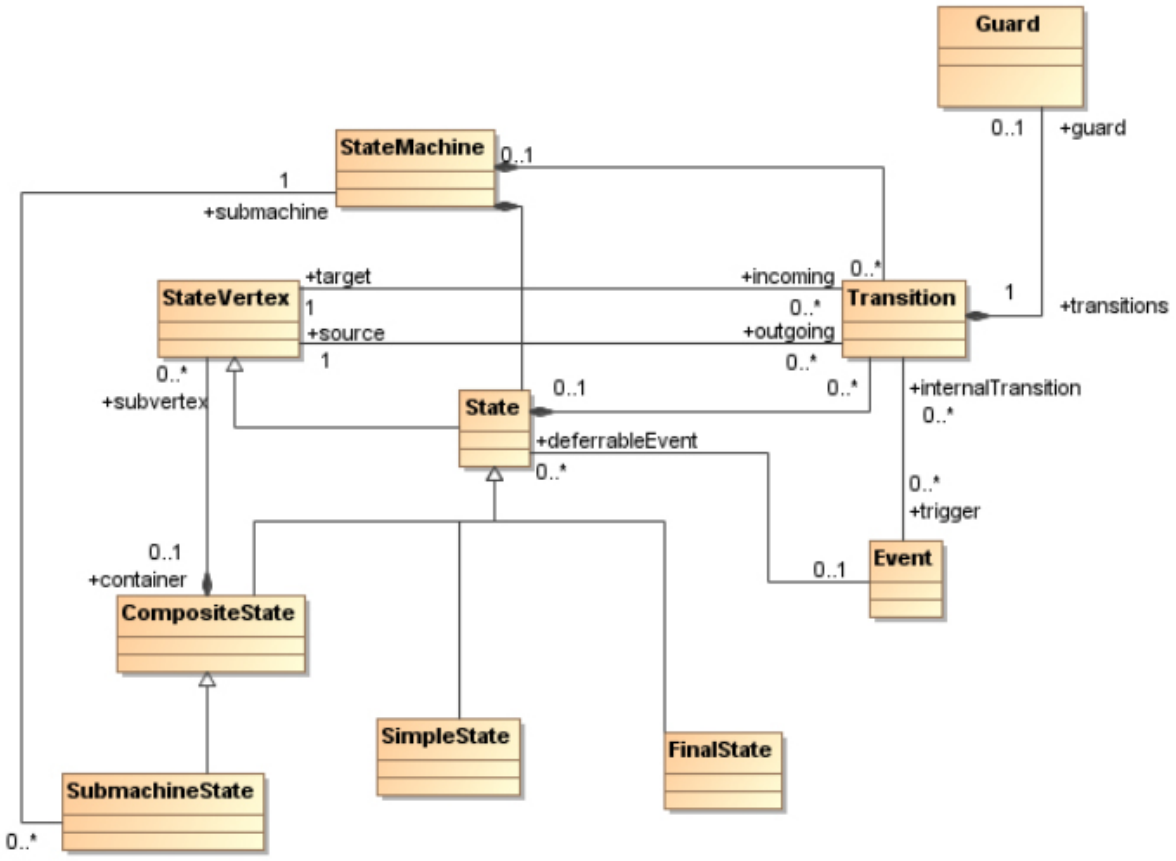
Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

Η UML προσφέρει τρεις μηχανισμούς επεκτασιμότητας:

1. **Tagged values**: Μέσω του μηχανισμού αυτού είναι δυνατό να προστεθούν αφηρημένες πληροφορίες στα στοιχεία του μοντέλου. Οι χρήστες μπορούν να προσθέσουν νέες ιδιότητες σε οποιοδήποτε στοιχείο του μοντέλου. Μία τέτοια «ετικέτα» είναι ουσιαστικά ένα ζεύγος λέξης κλειδιού – τιμής που περιγράφει μια συγκεκριμένη ιδιότητα. Κάθε στοιχείο του μοντέλου έχει ένα σύνολο από τέτοια ζεύγη τα οποία κατηγοριοποιούνται βάσει των λέξεων κλειδιών.

2. **Stereotypes**: Επιτρέπουν την ταξινόμηση των στοιχείων του μοντέλου. Τα stereotypes μπορούν να χρησιμοποιηθούν για να εισάγουν επιπρόσθετες κατηγοριοποιήσεις ανάμεσα στα στοιχεία του μοντέλου, οι οποίες δεν υποστηρίζονται ρητά από το μεταμοντέλο της UML. «Εφαρμόζοντας» ένα stereotype σε ένα στοιχείο του μοντέλου ουσιαστικά παράγουμε ένα εξειδικευμένο στοιχείο μοντέλου. Αυτός ο μηχανισμός επεκτασιμότητας μπορεί να συγκριθεί με την κληρονομικότητα.

3. **Constraints**: Επιτρέπουν την εισαγωγή νέων σημασιολογικών περιορισμώ. Με αυτό τον τρόπο είναι δυνατή ο ορισμός επιπρόσθετων περιορισμών που πρέπει να τηρούν τα στοιχεία.



εικόνα 10 *- διάγραμμα UML παράδειγμα 2*
source: *http://artemis-new.cslab.ece.ntua.gr:8080/jspui/bitstream/123456789/5936/1/PD2008-0014.pdf*

3. Μεθοδολογία ανάπτυξης

3.1. Μοντελοκεντρική ανάπτυξη

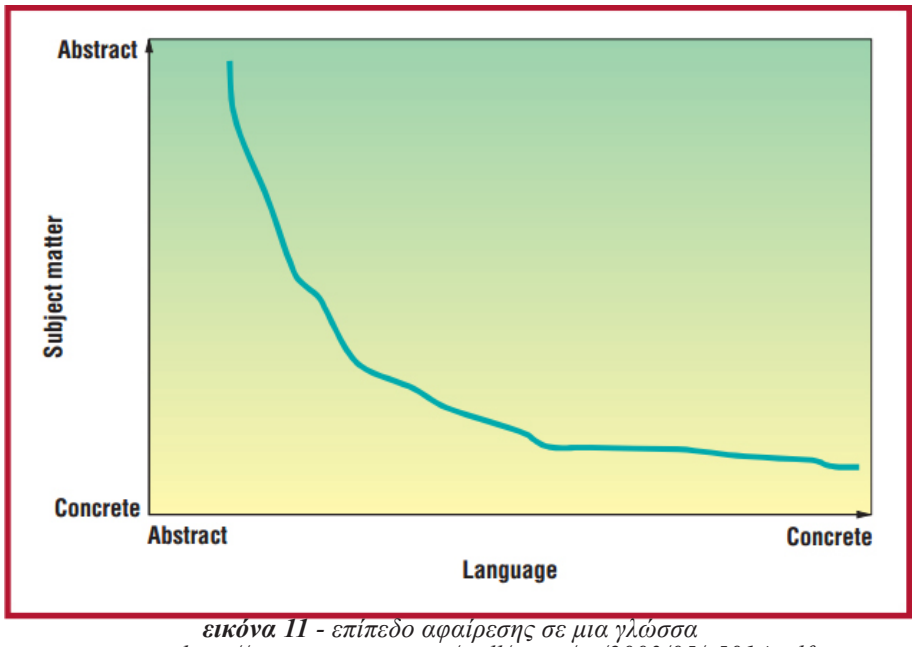
3.1.1. Εισαγωγή:

Μοντελοκεντρική Ανάπτυξη είναι απλώς η ιδέα πως μπορούμε να κατασκευάσουμε ένα μοντέλο ενός συστήματος που θα μπορεί στη συνέχεια να μετατραπεί σε κάτι συγκεκριμένο και απτό. ***[7]*** Με τον ορισμό αυτό είμαστε όλοι, αυτή τη στιγμή, εν δυνάμει χρήστες της μοντελοκεντρικής ανάπτυξης. Όταν γράφουμε ένα πρόγραμμα σε Smalltalk, Java ή C#, δεν περιμένουμε να εκτελέσει άμεσα. Περιμένουμε να μετατραπεί σε γλώσσα κάποιας μηχανής, που μπορεί να προκαλέσει το μοντέλο μας να κάνει τη δουλειά της.

Δεν σκέπτονται όλοι οι προγραμματιστές έτσι για μοντέλα σήμερα όμως. Πάρα πολύ συχνά εξισώνουν τα μοντέλα με απλές εικόνες, βγαλμένες από την ανάπτυξη πραγματικών συστημάτων, και είναι άσκοπα καθώς και βαριά στη διαδικασία. Πριν όμως καθιερώσουμε αυτή τη στενή αντίληψη, ας ρίξουμε μια βαθύτερη ματιά σε αυτό το μοντέλο, δηλαδή σε τι είναι και τι δεν είναι.

Ένα μοντέλο είναι ένα συνεκτικό σύνολο τυπικών στοιχείων που περιγράφουν κάτι (π.χ.: ένα σύστημα, μια τράπεζα, ένα τηλέφωνο, ή ένα τρένο) που χτίστηκε για κάποιο σκοπό, και που είναι δεκτικό σε μια συγκεκριμένη μορφή ανάλυσης, όπως:

- Επικοινωνία ιδεών μεταξύ ανθρώπων και μηχανών
- Έλεγχος πληρότητας
- Ανάλυση καταστάσεων
- Δοκιμή περίπτωσης παραγωγής
- Βιωσιμότητα δεικτών, όπως το κόστος και η εκτίμηση
- Πρότυπα
- Μετατροπή σε μια εφαρμογή



εικόνα 11 *- επίπεδο αφαίρεσης σε μια γλώσσα*
source: *http://www.computer.org/csdl/mags/so/2003/05/s5014.pdf*

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

Κάθε μοντέλο αντιμετωπίζει κάποια ζητήματα. Για παράδειγμα, θα μπορούσαμε να φτιάξουμε ένα μοντέλο τράπεζας, αγνοώντας την ασφάλεια και το περιβάλλον εργασίας, ή θα μπορούσε να διαμορφώσουμε ένα συνδυασμό αυτών των τομέων. Επιλέγουμε ποιο αντικείμενο αξίζει να περιληφθεί και ποιο να αγνοηθεί αρχικά, αν και πάλι θα μπορούσε να χρειαστεί στη συνέχεια αυτά τα αντικείμενα να συνεργαστούν. Όταν το θέμα ενός μοντέλου έχει ένα υψηλό βαθμό αφαίρεσης (abstraction), το μοντέλο είναι πιο κοντά στην τελική γλώσσα του χρήστη, όπου υπάρχει ένα μικρό χάσμα μεταξύ ενός non-computer expert και του μοντέλου.

Επιπλέον, εκφράζουμε ένα μοντέλο σε μια γλώσσα όπου υπάρχει σε ένα επίπεδο αφαίρεσης. Ένα μοντέλο γραμμένο στη γλώσσα C - μοντελοποίησης, θα αγνοήσει την πραγματοποίηση των κλήσεων λειτουργίας και έκφρασης, αφήνοντας τα CPU - προσανατολισμένα θέματα, όπως το register allocation στον compiler ή σε μια εικονική μηχανή διερμηνέα, που προσθέτει πραγματοποίηση κατά το χρόνο εκτέλεσης. Ομοίως, ένα μοντέλο που εκφράζεται σε UML θα αγνοήσει την πραγματοποίηση των ενώσεων, αφήνοντας εκείνες αποφάσεις σε ένα μοντέλο compiler ή σε ανθρώπινο σχεδιασμό. Το σχήμα 1 απεικονίζει αυτές τις δύο διαστάσεις: Επίπεδο Αφαίρεσης της Γλώσσας και ο Βαθμός της Αφαίρεσης του Θέματος υπό μελέτη. Ο άξονας θέμα έχει μια ανεστραμμένη κλίμακα, η οποία οδηγεί σε μη τακτοποιημένη καμπύλη με μοντέλα ανάλυσης στην κορυφή και το σχεδιασμό μοντέλων χαμηλότερα.

3.1.2. Μοντελοκεντρική αρχιτεκτονική:

Η μοντελοκεντρική αρχιτεκτονική (**Model Driven Architecture – MDA**) είναι μία πρόταση του OMG για τη διαδικασία ανάπτυξης λογισμικού. ***[6]*** Η κύρια ιδέα της MDA είναι η αξιοποίηση των μοντέλων στην ανάπτυξη εφαρμογών λογισμικού. Σύμφωνα με την MDA, η διαδικασία ανάπτυξης λογισμικού κατευθύνεται από τη διαδικασία μοντελοποίησης του συστήματος λογισμικού. Ο κύκλος ανάπτυξης της MDA φαίνεται στην *Εικόνα 12* και δεν διαφέρει πολύ από τον κλασικό κύκλο ανάπτυξης. Η διαφορά έγκειται στο ότι σε κάθε φάση παράγονται μοντέλα σε μορφή που μπορούν να γίνουν κατανοητά από υπολογιστές.

Ανεξάρτητο από τη πλατφόρμα μοντέλο:

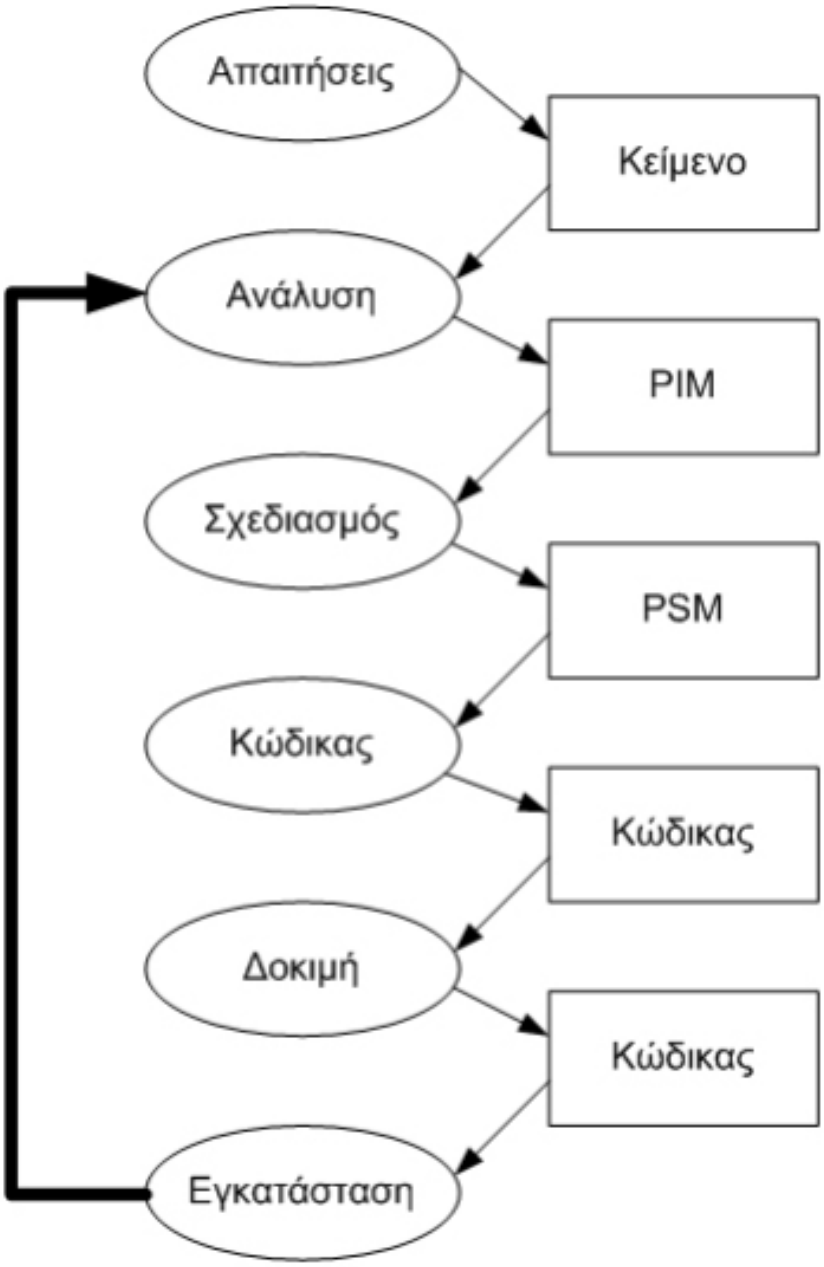
Το ανεξάρτητο από τη πλατφόρμα μοντέλο (Platform Independent Model – PIM) ορίζει ένα μοντέλο υψηλού επιπέδου ανεξάρτητο από τη τεχνολογία στην οποία θα γίνει η υλοποίησή του. Το μοντέλο αυτό περιγράφει το σύστημα λογισμικού που θα αναπτυχθεί με τον τρόπο που κρίνεται καλύτερος για τη συγκεκριμένη εφαρμογή, ανεξάρτητα της τεχνολογίας που τελικά θα χρησιμοποιηθεί.

Μοντέλο ειδικά για τη πλατφόρμα:

Στο επόμενο βήμα, το PIM μετατρέπεται σε ένα μοντέλο ειδικά για την πλατφόρμα (Platform Specific Model - PSM). Το PSM ορίζει τις δομές που θα χρησιμοποιηθούν στην πλατφόρμα που θα πραγματοποιηθεί η υλοποίηση. Ένα PIM μπορεί να μετατραπεί σε παραπάνω από ένα PSM, αν χρησιμοποιούνται παραπάνω από μία τεχνολογικές πλατφόρμες για την εφαρμογή.

Κώδικας:

Το τελευταίο βήμα κατά τη διαδικασία ανάπτυξης είναι η μετατροπή του PSM σε κώδικα. Μιας και το PSM είναι άμεσα συνδεδεμένο με την τεχνολογία υλοποίησης, ο μετασχηματισμός αυτός είναι άμεσος.



***εικόνα 12** - Model Driven Architecture*
***source:** http://artemis-new.cslab.ece.ntua.gr:8080/jspui/bitstream/123456789/5936/1/PD2008-0014.pdf*

Από τα παραπάνω γίνεται φανερό ότι η ανάπτυξη μιας εφαρμογής χρησιμοποιώντας την μοντελοκεντρική αρχιτεκτονική προσθέτει επιπλέον αφαιρετικότητα στον τρόπο σχεδιασμού μέσω του PIM. Το πρόβλημα που εμφανίζεται είναι ότι ο μετασχηματισμός από ένα PIM σε ένα PSM και από PSM σε κώδικα είναι χρονοβόρα διαδικασία. Ωστόσο αυτή η διαδικασία μπορεί να αυτοματοποιηθεί π.χ. με εργαλεία που παράγουν κώδικα. Η εισαγωγή της μοντελοκεντρικής αρχιτεκτονικής στη διαδικασία ανάπτυξης λογισμικού

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

μπορεί να γίνει με πολλούς διαφορετικούς τρόπους και σε πολλά διαφορετικά σημεία. Εκτός από αυτά τα ολοκληρωμένα εργαλεία, εφαρμογές της MDA υπάρχουν σε πολλές άλλες περιπτώσεις, όπου το μοντέλο υπάρχει σε μορφή XMI και οι μετασχηματισμοί είναι γραμμένοι σε XSL.

3.1.3. MDA: προσόν ή εμπόδιο?

Ένα μοντέλο δεν χρειάζεται να είναι πάντα πλήρης. ***[7]*** Συχνά, ένα γραφικό μοντέλο τελικά αποκλείει μέρος κώδικα, αν και σε καμία περίπτωση δεν σημαίνει απαραίτητα ότι η UML είναι μια ολοκληρωμένη, υπολογιστικά, γλώσσα. Επιπλέον, ένα μοντέλο έχει πολλαπλές όψεις, μερικές εκ των οποίων αποκαλύπτονται τελικά. Για παράδειγμα, μπορούμε να εκθέσουμε μεμονωμένα συνεργαζόμενες μηχανές σε ένα διάγραμμα statechart, ή μπορούμε να ξεχωρίσουμε τις συνεργασίες τους, άμεσα, χρησιμοποιώντας ένα διάγραμμα ακολουθίας. Σαφώς, οποιοδήποτε σύνολο διαγραμματικών απόψεων πρέπει να είναι συνεπές με το υποκείμενο μοντέλο, του οποίου είναι σύνολο προβολών. Η πληρότητα και ένας υψηλός βαθμός αφαίρεσης δεν ισοδυναμεί με ασάφεια. Δεν είναι, αλλά και δεν μπορούν οπωσδήποτε να είναι εκτελέσιμα ή να είναι ακόμη και τυπικά όλα τα μοντέλα, αλλά εκείνα που είναι και μπορούν, επωφελούνται προφανώς από την αυτοματοποίηση αυτή.

Χρησιμοποιούμε τα μοντέλα για να αυξήσουμε την παραγωγικότητα λοιπόν. Είναι προσοδοφόρο να γράψει κανείς μια γραμμή Java από το να γράψει 10 γραμμές γλώσσας assembly. Ομοίως, ή με το επιχείρημα αυτό, είναι φθηνότερο να χτίσουμε ένα γραφικό μοντέλο σε UML γλώσσα, από το να γράψουμε σε Java επομένως. Θα αφήσουμε την ανησυχία αυτή για την ώρα ώστε να συνεχίσουμε παρακάτω.

Η ανησυχία μας, περισσότερο, λαμβάνει χώρα εδώ ακριβώς όπου κάποιοι υποστηρίζουν ότι τα μοντέλα προσφέρουν περισσότερα εμπόδια παρά βοήθεια. Μερικοί υποστηρικτές των extreme διαδικασιών υποστηρίζουν ότι ένα μοντέλο χρησιμοποιείται συχνά για να υποδεικνύει ένα σχεδιάγραμμα, που ενεργεί ως διεπαφή μεταξύ των προγραμματιστών. Επιπλέον, υποστηρίζουν ότι αυτή η διεπαφή έχει αδυναμίες:

- Οι ανησυχίες του αναλυτή δεν είναι και οι ανησυχίες του προγραμματιστή. Τόσο πολύ διαφέρουν, έτσι ώστε η διαγραμματική ανάλυση έχει απλώς συμβουλευτικό χαρακτήρα, και πιθανόν κακό στην τελική.
- Η γλώσσα στην οποία οι μοντελιστές κατασκευάζουν τα σχέδια ορίζεται συχνά ως ill - defined και δύσκολα μεταφράζεται αξιόπιστα σε κώδικα.
- Τα σχεδιαγράμματα βρίσκονται out-of-date πριν ακόμα τελειώσουν.
- Οι μοντελιστές συχνά σκοπεύουν με αυτά τα σχέδια να προβλέψουν το απρόβλεπτο, την δημιουργική πράξη του να εφεύρει κανείς αφαιρέσεις στον κώδικα.

Επιπλέον, η μοντελο-κεντρική ανάπτυξη προσφέρει τη δυνατότητα για αυτόματη μεταμόρφωση του υψηλού επιπέδου αφηρημένων μοντέλων. Ζήτημα εφαρμογών που υπόκεινται σε συστήματα λειτουργίας. Σε αυτό το ζήτημα, ο Bran Selic ("The Pragmatics of Model-Driven Software Development") υποστηρίζει ότι η τεχνολογία μοντελοποίησης έχει ωριμάσει σε σημείο που μπορεί να προσφέρει σημαντική μόχλευση

σε όλες τις πτυχές της ανάπτυξης λογισμικού. Υποστηρίζει επίσης ότι σε έναν συνεχώς αυξανόμενο αριθμό τομέων εφαρμογής, μπορεί κανείς να δημιουργήσει ένα μεγάλο μέρος του κώδικα της εφαρμογής απευθείας από τα μοντέλα.

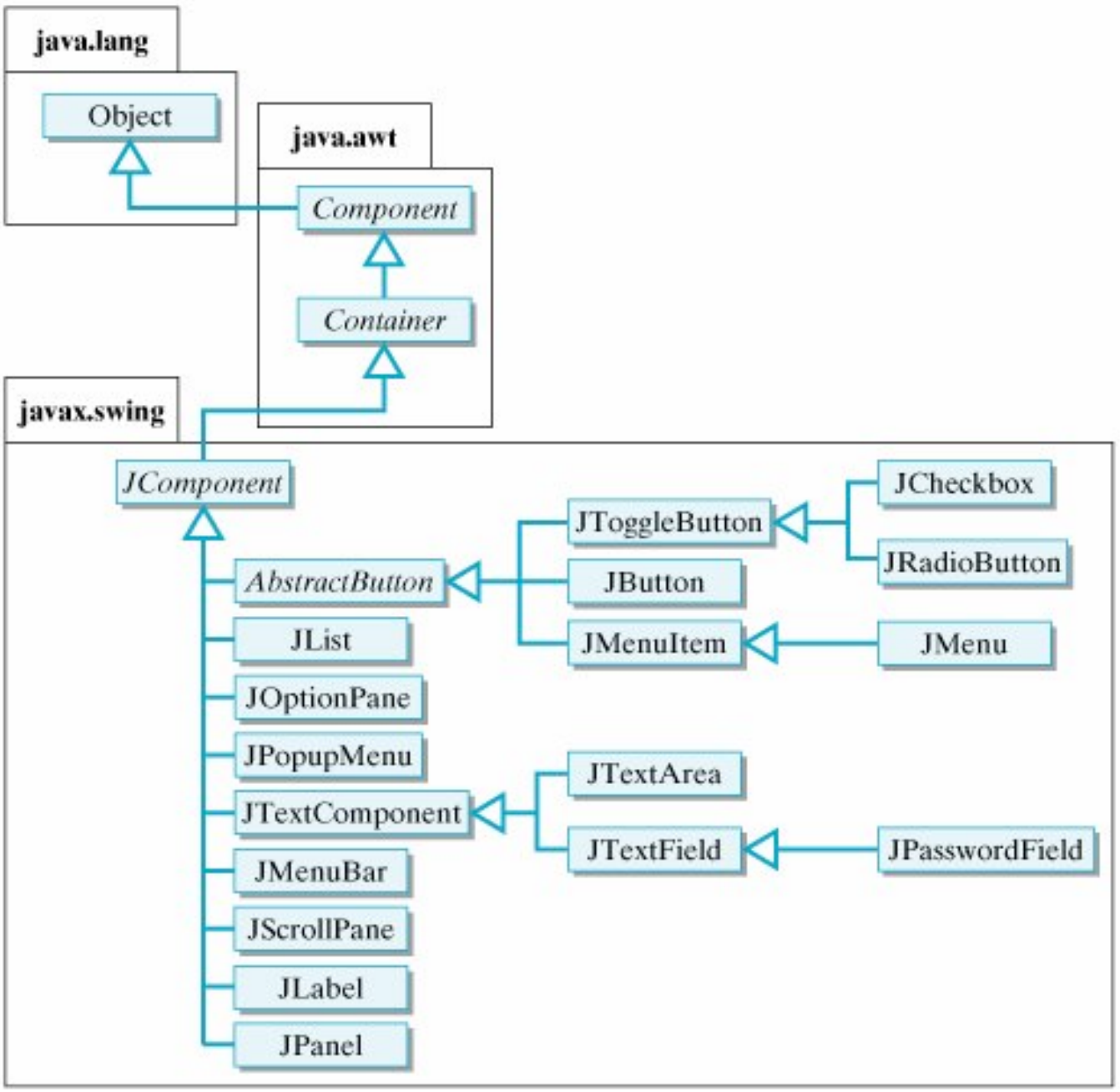
Η αυτοματοποίηση δεν αναιρεί την αναγκαιότητα για δημιουργικότητα. Αντίθετα, επισημοποιεί τις υπάρχουσες λύσεις και αυξάνει το επίπεδο στο οποίο μπορούμε να εφαρμόσουμε τη δημιουργικότητα, δίνοντας έτσι στον προγραμματιστή περισσότερη δύναμη. Με τη σειρά μας, θα πρέπει τώρα να αναρωτηθούμε πώς μπορούμε να εμπνεύσουμε προγραμματιστές που να πιστεύουν σε ένα επίπεδο αφαίρεσης πάνω από τα σημερινά δεδομένα που ορίζει ο προγραμματισμός ή η τεχνολογία προγραμματισμού, όπως την ξέρουμε.

3.2. Το Java Swing

3.2.1. Γραφικές διεπαφές τύπου Swing (Swing GUIs):

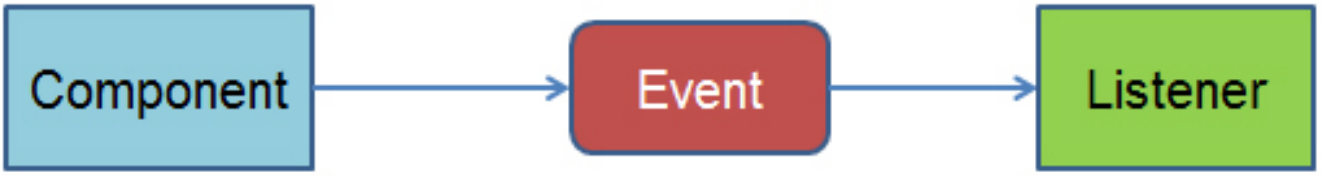
Η Swing είναι η βιβλιοθήκη της Java για τον προγραμματισμό **GUIs** (Graphical User Interfaces), και πιο γενικά για την χρήση γραφικών. ***[8] & [9]*** Τα πακέτα που αρχίζουν με το πρόθεμα **javax.swing** παρέχουν ευέλικτα και ισχυρά εργαλεία ανάπτυξης GUI. Η μετεξέλιξη του **AWT** (Abstract Window Toolkit) το οποίο ήταν το πρώτο αλλά όχι τόσο επιτυχημένο πακέτο της Java για GUI. Το πακέτο javax.swing αναπτύχθηκε κυρίως λόγω των ανεπαρκειών του Abstract Windows Toolkit (AWT). Παραδείγματος χάριν, η κλάση **JButton** που ορίζεται σε αυτό, υπερτερεί της **AWT Button** κλάσης επιτρέποντας την ύπαρξη όχι μόνο απλού κείμενου, αλλά και εικόνων στα κουμπιά.

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389



εικόνα 13 *- UI's και Java Swing*
source: *http://www.slideshare.net/itc73/3-12015850*

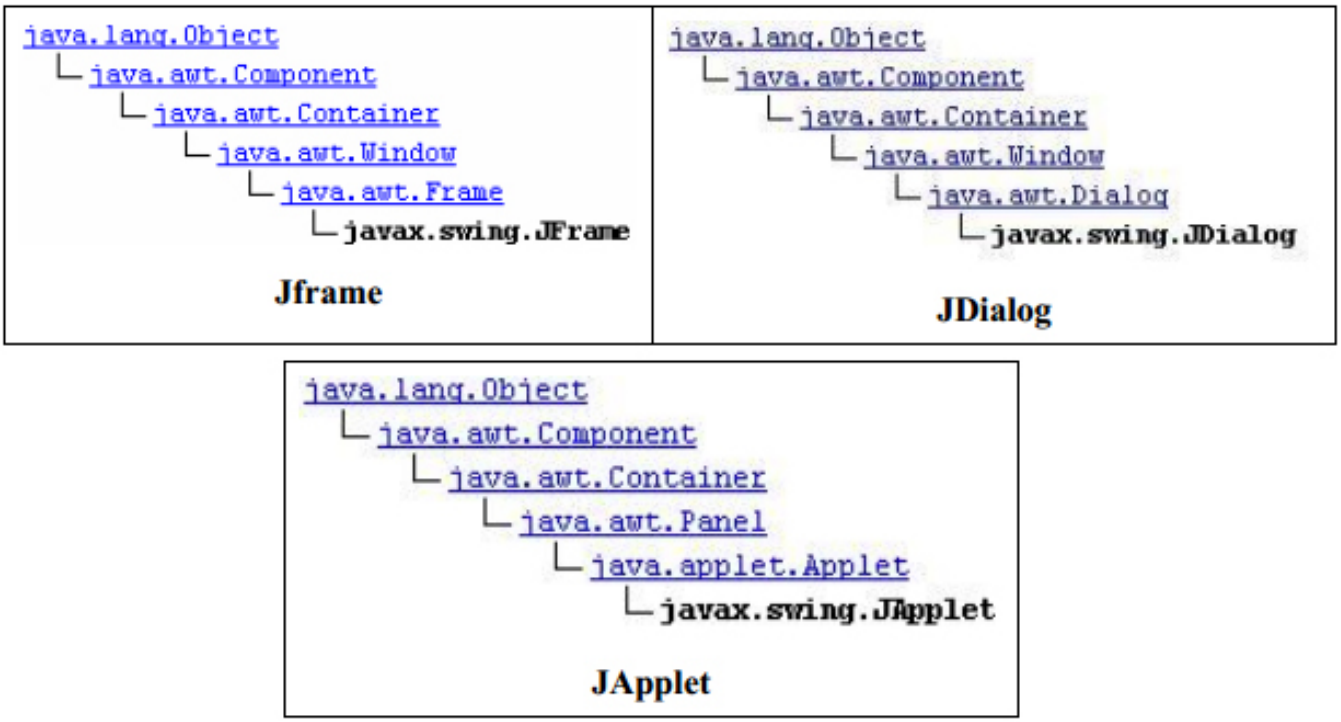
Στην Swing βιβλιοθήκη ένα GUI αποτελείται από πολλά στοιχεία - συστατικά (**components**) π.χ.: παράθυρα, κουμπιά, μενού, κουτιά εισαγωγής κειμένου, κλπ. Τα components αυτά πυροδοτούν συμβάντα, π.χ.: το πάτημα ενός κουμπιού, η εισαγωγή κειμένου, η επιλογή σε ένα μενού, κλπ. Τα συμβάντα αυτά τα χειρίζονται τα αντικείμενα - ακροατές, που έχουν ειδικές μεθόδους γι' αυτά. Τι γίνεται όταν πατάμε ένα κουμπί, όταν κάνουμε μια επιλογή, κλπ. Όλο το πρόγραμμα κυλάει ως μια αλληλουχία από συμβάντα και τον χειρισμό των ακροατών.



***εικόνα 14** - πυροδότηση συμβάντων στο Swing*
***source:** http://www.cs.uoi.gr/~tsap/teaching/cse205/lectures/oop25.pptx*

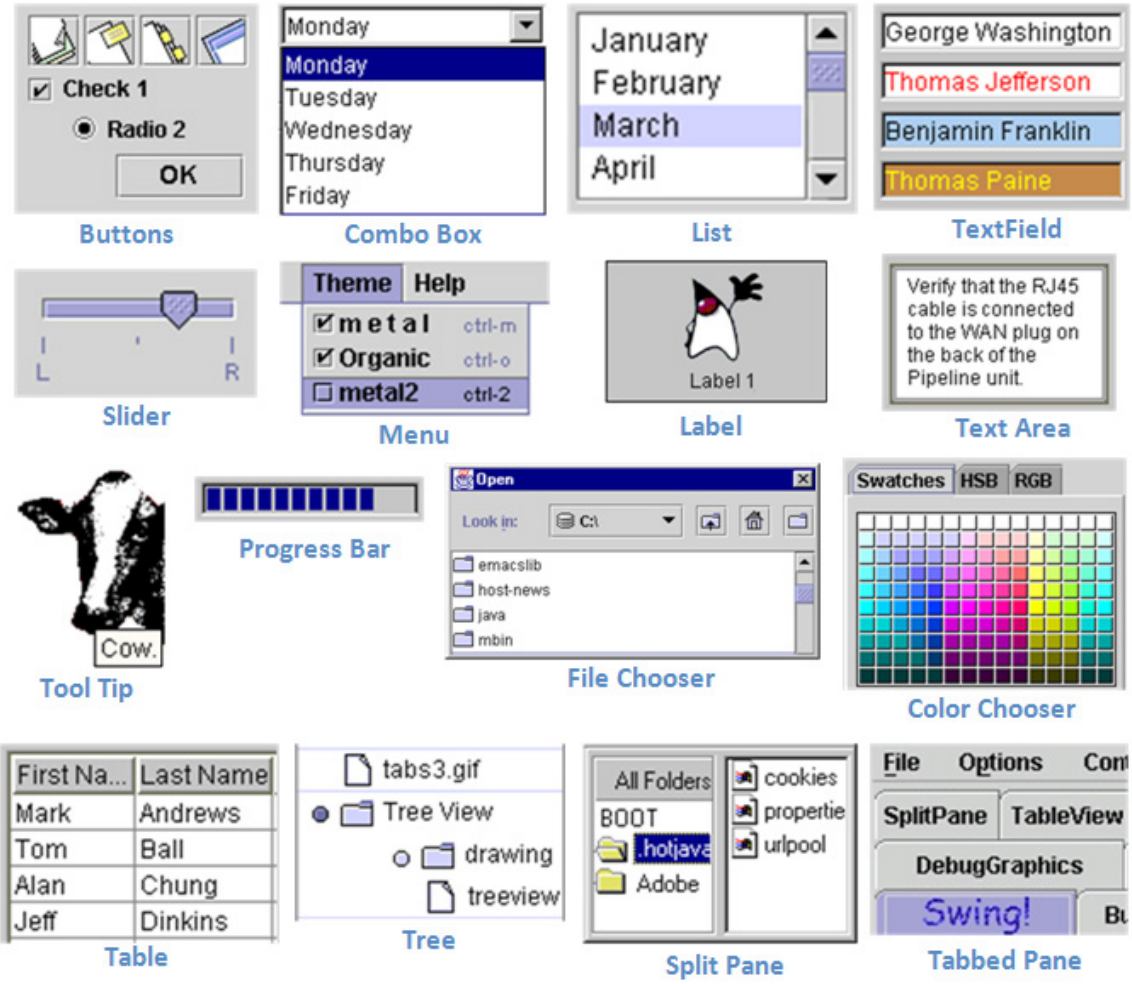
3.2.2. Top-Level Swing Containers και Swing Components:

Κάθε γραφική διεπαφή τύπου Swing πρέπει να έχει τουλάχιστον ένα **top-level Swing container**. ***[8]*** Ένα top-level Swing container παρέχει την απαραίτητη υποστήριξη που χρειάζονται τα Swing components για την εμφάνισή τους και την διαχείριση των γεγονότων που αυτά παράγουν. Υπάρχουν τρία top-level Swing containers: το **JFrame**, το **JDialog**, και (για applets) το **JApplet**. Κάθε JFrame αντικείμενο δημιουργεί ένα κύριο γραφικό παράθυρο, κάθε JDialog αντικείμενο δημιουργεί ένα δευτερεύον παράθυρο (δηλαδή παράθυρο που εξαρτάται από κάποιο άλλο παράθυρο). Κάθε JApplet αντικείμενο δημιουργεί την περιοχή εμφάνισης ενός applet στο παράθυρο του Web Browser. Η ιεραρχία των top-level Swing containers δίνεται παρακάτω:



***εικόνα 15** - ιεραρχία top-level Swing containers*
***source:** http://www.medialab.ntua.gr/education/MultimediaTechnology/JavaNotes/files/8.%20Java_swing.pdf*

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389



εικόνα 16 *- τα components του Swing*
source: *http://www.cs.cornell.edu/courses/CS2110/2014fa/L15-GUIS_Layout/cs2110fa14GUILayout-6up.pdf*

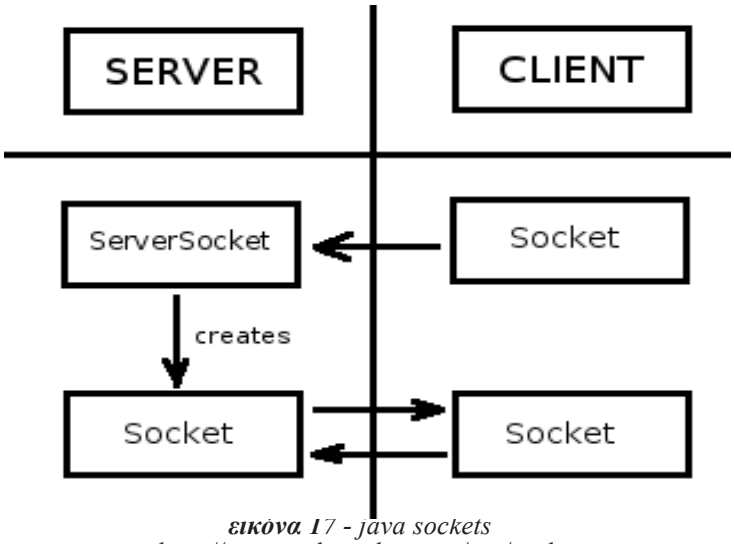
Η χρήση των υπολοίπων Swing Components είναι παρόμοια. Συνήθως, όταν υλοποιείται μια GUI εφαρμογή σε Swing, δημιουργούμε ένα JFrame αντικείμενο και επιλέγουμε συγκεκριμένο πλάνο ή σχέδιο (**layout**) για αυτό. Κατόπιν τοποθετούμε ένα ή περισσότερα JPanels στο JFrame. Τα JPanels διαθέτουν επίσης επιλογές layout όπως και τα JFrames. Στη συνέχεια, μπορούμε να προσθέτουμε και άλλα αντικείμενα τύπου Component.

3.3. Πρωτόκολλο TCP

3.3.1. Εισαγωγή:

Η υλοποίηση του πρωτοκόλλου μεταφοράς TCP είναι προσανατολισμένη για ενσύρματα δίκτυα δεδομένων παρέχοντας μια ασφαλή μεταφορά δεδομένων μμεταξύ σταθερών σταθμών επικοινωνίας. ***[10]*** Η εφαρμογή του TCP έχει επεκταθεί σε κινητούς σταθμούς παρέχοντας την αντίστοιχη υπηρεσία με κάποιους όμως περιορισμούς στην απόδοση. Το πρωτόκολλο TCP είναι ένα πρωτόκολλο επιπέδου Μεταφοράς (Transport layer) σχεδιασμένο να παρέχει connection-oriented υπηρεσίες.

Το TCP και το IP είναι ο πλέον συνηθέστερος σήμερα χρησιμοποιούμενος συνδυασμός πρωτοκόλλων για την επίτευξη επικοινωνίας στο Διαδίκτυο. Όπως παρακάτω θα συναντήσουμε, από την επεξήγηση κώδικα που γίνεται στα επόμενα κεφάλαια, εμείς θα ασχοληθούμε με τα **Java sockets**. Η ομορφιά των Java sockets είναι πως δεν χρειάζεται κανείς να έχει μεγάλες γνώσεις για τις λεπτομέρειες του TCP. Το TCP (Transmission Control Protocol) είναι ένα πρωτόκολλο ελέγχου μετάδοσης και είναι ένα τυπικό πρωτόκολλο διαβίβασης δεδομένων, με επιβεβαίωση λήψης δεδομένων. Για να ξεκινήσει μια συνεδρία TCP, ένας διακομιστής και ένας πελάτης απαιτούνται. Ο διακομιστής έχει συσταθεί για να ακούει σε ένα συγκεκριμένο **port**. Έπειτα, περιμένει και δεν κάνει τίποτα μέχρι ο πελάτης να προσπαθεί να συνδεθεί με αυτό το port. Αν όλα πάνε καλά, η σύνδεση είναι επιτυχής και διακομιστής και πελάτης τρέχουν ένα instance της κλάσης **Socket**. Και από τις δύο οντότητες (Server & Client), ένα ρεύμα εισόδου (**Inputstream**) και ένα ρεύμα εξόδου(**Outstream**) μπορεί να ληφθεί. Όλη η επικοινωνία γίνεται μέσω αυτών των ρευμάτων.

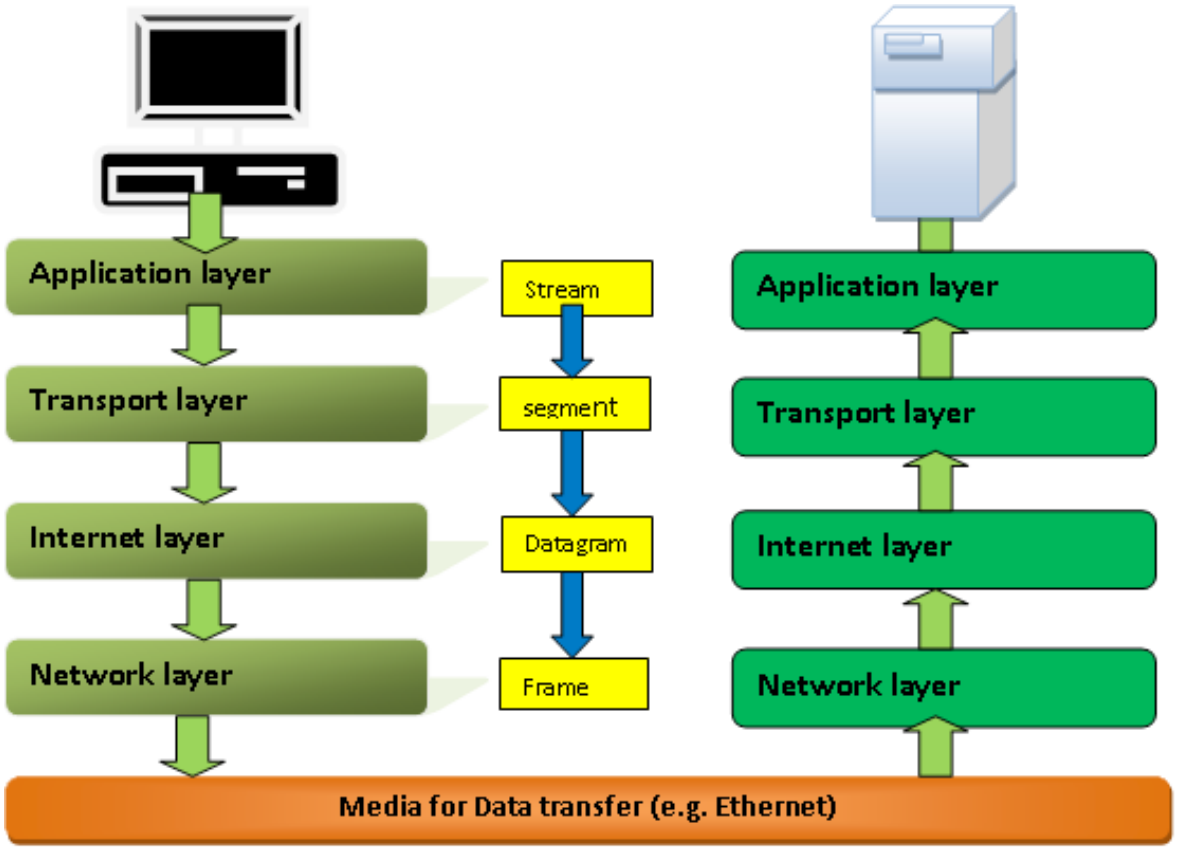


***εικόνα 17** - java sockets*
***source:** http://www.techntechie.com/tag/socket-server*

3.3.2. Το Μοντέλο TCP αναλυτικότερα:

Το μοντέλο TCP είναι ειδικά σχεδιασμένο, ώστε να παρέχει μία αξιόπιστη απ' άκρο σ' άκρο ροή δεδομένων, πάνω σε ένα αναξιόπιστο διαδίκτυο. Ένα διαδίκτυο διαφέρει από ένα δίκτυο συγκεκριμένης μορφής, διότι απαρτίζεται από τμήματα που έχουν διαφορετικές παραμέτρους όπως τοπολογία, εύρος καναλιού (bandwidth), καθυστερήσεις, μέγεθος πακέτων κ.α. Το TCP προσαρμόζεται δυναμικά στις ιδιαιτερότητες του διαδικτύου και εμφανίζεται ανθεκτικό στην αντιμετώπιση διαφόρων τέτοιων προβλημάτων. Κάθε μηχάνημα που υποστηρίζει το TCP έχει και μία TCP οντότητα επιπέδου μεταφοράς είτε ως πρόγραμμα είτε ως μέρος του kernel η οποία διαχειρίζεται τις TCP ροές και ρυθμίζει την διεπαφή με το κατώτερο επίπεδο IP. Μία TCP οντότητα δέχεται μία ροή δεδομένων χρήστη από τοπικές διεργασίες, τη σπάει σε τμήματα μέχρι 64K (στην πράξη, συνήθως μέχρι 1500 byte) και στέλνει κάθε τμήμα σαν ξεχωριστό πακέτο IP.

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389



εικόνα 18 *- μοντέλο TCP/IP*
source: *http://kmlstudent.blogspot.gr/2012/07/tcpip-model-explained-with-diagram.html*

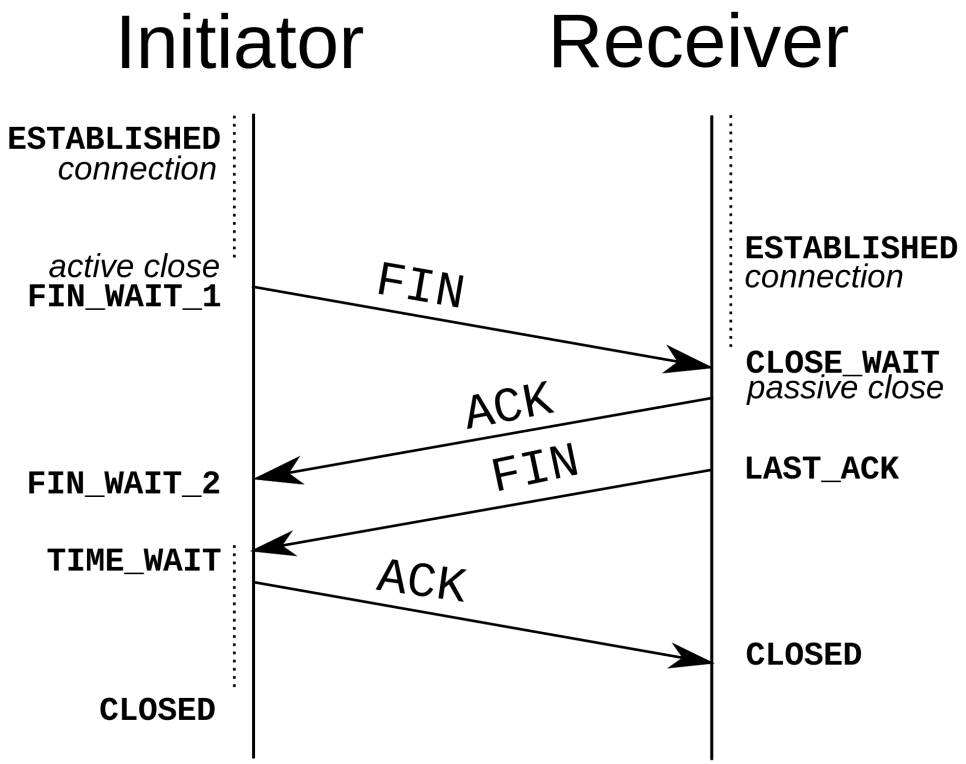
Όταν πακέτα IP που περιέχουν δεδομένα TCP καταφθάνουν σε ένα μηχάνημα τότε αυτά παραδίδονται στην οντότητα του TCP όπου επανασχηματίζει την αρχική ροή δεδομένων. Το επίπεδο δικτύου δεν παρέχει καμία εγγύηση, ότι τα πακέτα IP θα παραδοθούν σωστά και στην σειρά με την οποία μμεταδόθηκαν. Έτσι το TCP θα πρέπει να φροντίσει για επαναμεταδόσεις λανθασμένων ή χαμένων πακέτων όπως και για την αναδιάταξη των μηνυμάτων στη σωστή σειρά.

Κάθε byte σε μία TCP σύνδεση έχει ένα μοναδικό 32άμπιτο αριθμό ακολουθίας (**sequence number**). Αυτοί οι αριθμοί ακολουθίας χρησιμοποιούνται για τις επιβεβαιώσεις (acknowledgement) και για τον μηχανισμό του παραθύρου. Οι TCP οντότητες του αποστολέα και του δέκτη ανταλλάσσουν δεδομένα με τη μορφή πακέτων (segment). Ένα πακέτο αποτελείται από μία επικεφαλίδα σταθερού μεγέθους 20-byte ακολουθούμενο από μηδενικά ή περισσότερα byte δεδομένων. Το TCP αποφασίζει πόσο θα είναι το μέγεθος κάθε πακέτου. Μπορεί είτε να καταχωρήσει δεδομένα από πολλά γραψίματα σε ένα πακέτο είτε να σπάσει τα δεδομένα από ένα γράψιμο σε πολλά πακέτα. Δύο περιορισμοί καθορίζουν το μέγεθος κάθε πακέτου. Ο πρώτος περιορισμός είναι ο διαθέσιμος χώρος του IP πακέτου ο οποίος δεν πρέπει να ξεπερνά τα 65.535 byte. Ο δεύτερος αφορά την μέγιστη μονάδα μεταφοράς (Maximum Transfer Unit – **MTU**) ενός δικτύου όπου μέσα σε αυτή θα πρέπει να χωράει κάθε πακέτο του TCP. Αν ένα TCP πακέτο καλύπτει τον περιορισμό του MTU ενός δικτύου τότε ενθυλακώνεται στο IP πακέτο χωρίς να τεμαχιστεί ενώ όταν υπερβαίνει τον περιορισμό MTU τότε τεμαχίζεται σε περισσότερα από ένα πακέτα. Ο βασικός μηχανισμός που εφαρμόζουν οι οντότητες του TCP είναι το κυλιόμενο παράθυρο. Όταν ο αποστολέας μεταδίδει ένα πακέτο αρχίζει ένας χρονομετρητής. Όταν το πακέτο φθάσει στον προορισμό, η οντότητα TCP του δέκτη στέλνει ένα πακέτο στον αποστολέα, συμπεριλαμβάνοντας σε αυτό έναν αριθμό

επιβεβαίωσης ίσο με τον επόμενο αριθμό της ακολουθίας δεδομένων που περιμένει να δεχτεί. Εάν ο χρονομετρητής του αποστολέα εκπνεύσει προτού έρθει η επιβεβαίωση τότε το πακέτο επαναμεταδίδεται. Στην περίπτωση όπου κάποιο πακέτο τεμαχιστεί από έναν δρομολογητή δικτύου που υποστηρίζει μικρότερα MTU από αυτά των διασυνδεμένων δικτύων, τότε είναι πιθανόν κάποια πακέτα να καθυστερήσουν να παραδοθούν στον δέκτη ή ακόμα και να χαθούν. Έτσι λοιπόν κάποια πακέτα που φθάνουν στον δέκτη με λάθος σειρά δεν μπορούν να επιβεβαιωθούν επειδή εκείνα που προηγούνται δεν έχουν ακόμα παραδοθεί. Τα πακέτα μπορεί ακόμα να καθυστερήσουν τόσο, ώστε ο χρονομετρητής να προκαλέσει την επαναμετάδοση αυτού του πακέτου. Εάν ένα πακέτο που επαναμεταδίδεται δρομολογηθεί από ένα άλλο μονοπάτι από αυτό του αρχικού πακέτου και τεμαχιστεί διαφορετικά, τότε τμήματα αυτού πακέτου και του αρχικού παραδίδονται σποραδικά στον δέκτη, απαιτώντας έτσι μία προσεκτική διαχείριση ώστε να ανακτηθεί το αρχικό πακέτο. Τέλος, λόγω του ότι το Internet δομείται από την διασύνδεση ποικίλων δικτύων, ενδέχεται κάποιο πακέτο να διαπερνάει, περιστασιακά κατά τη διαδρομή του μέσα από δίκτυα που παρουσιάζουν συμφόρηση.

3.3.3. Κύριες λειτουργίες TCP:

- Διευθυνσιοδότηση.
- Πολυπλεξία.
- Υλοποίηση (establishment) και διαχείριση σύνδεσης
  - ► Χειρισμός πολιτικών μετάδοσης / επαναμετάδοσης.
  - ► Έλεγχος ροής και ενταμίευση (buffering).
- Απόλυση σύνδεσης (connection release).



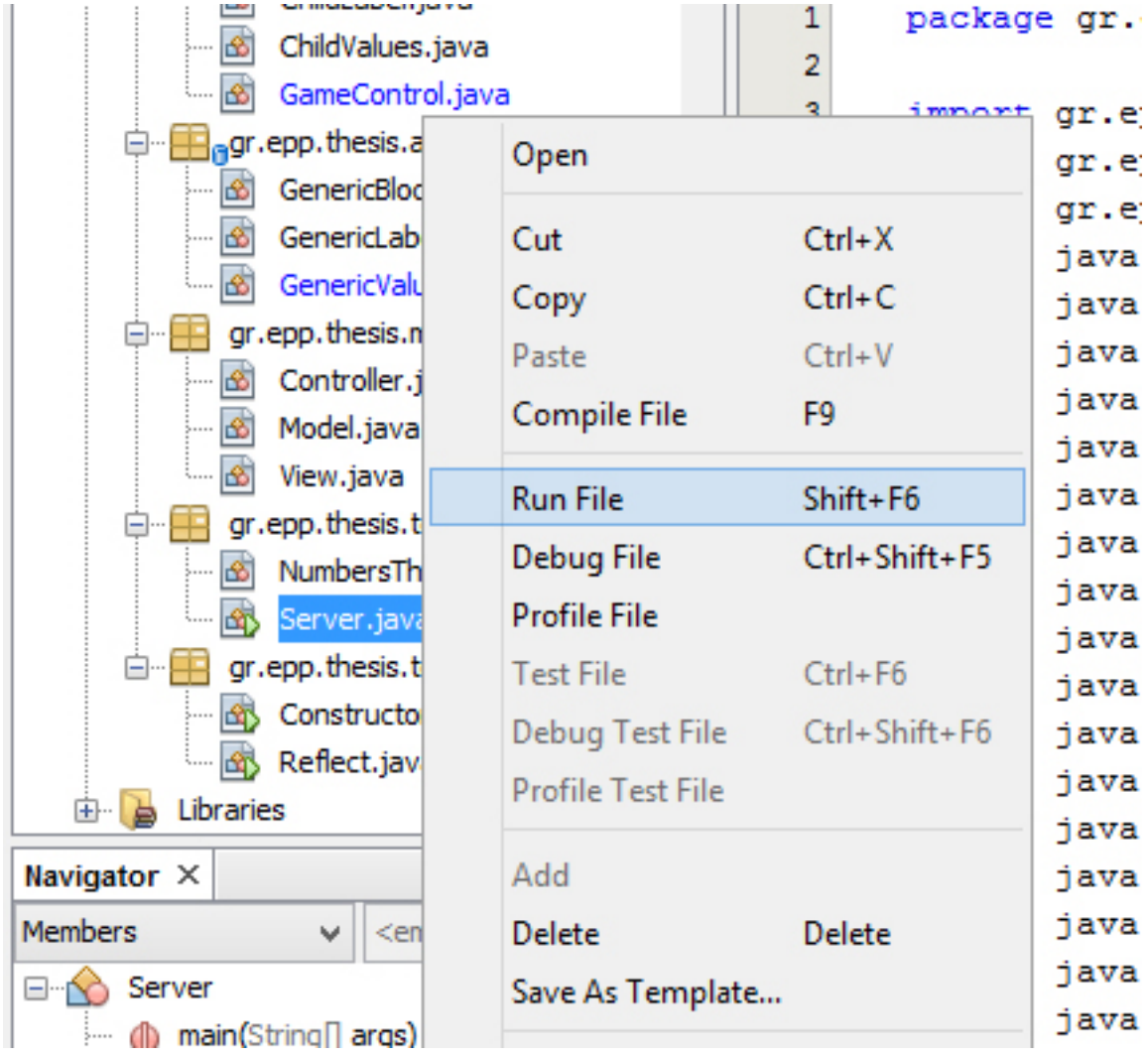
εικόνα 19 *- κύριες λειτουργίες TCP*
source: *http://en.wikipedia.org/wiki/TCP*

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

4. Ανάπτυξη συνεργατικού παιγνίου: Η Ναυμαχία

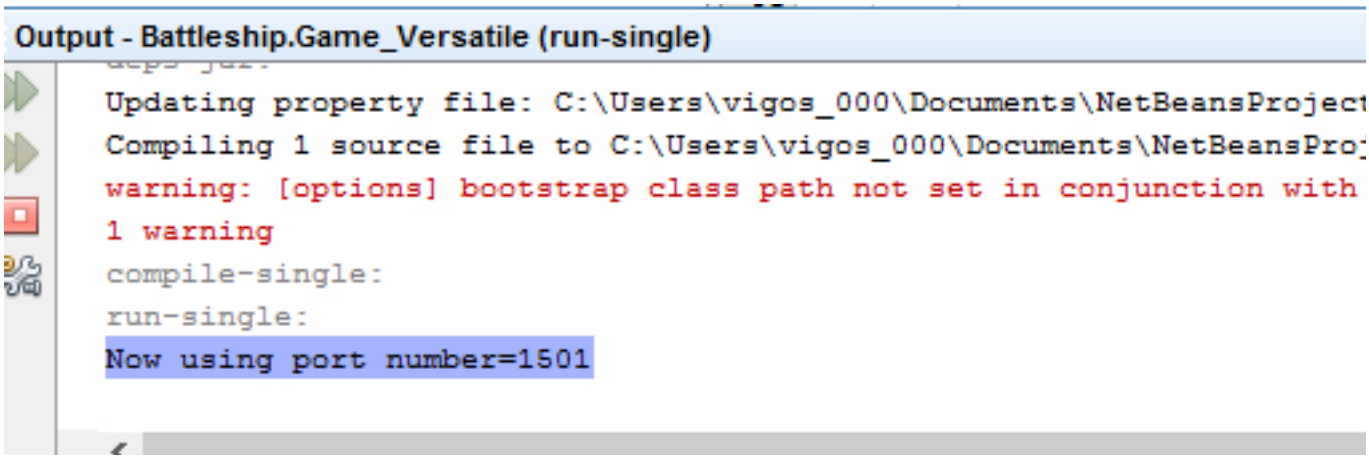
4.1. Εισαγωγή:

Θέλουμε σε αυτό το σημείο να εξετάσουμε μια περίπτωση του παιχνιδιού, από την αρχή του ως το τέλος, ούτως ώστε να μπούμε στο νόημα και να καταλάβουμε σιγά σιγά πως λειτουργεί. Θα εξετάσουμε την περίπτωση όπου συνδέεται στο παιχνίδι ένας ενήλικας και το παιδί του. (π.χ. πατέρας και γιος). Η εφαρμογή λειτουργεί με TCP, άρα και μπορούν να αναμετρηθούν από τους δικούς τους προσωπικούς υπολογιστές, μιας και η εφαρμογή μας τρέχει server όπου υποδέχεται **clients** (παίκτες) σε συγκεκριμένο port.



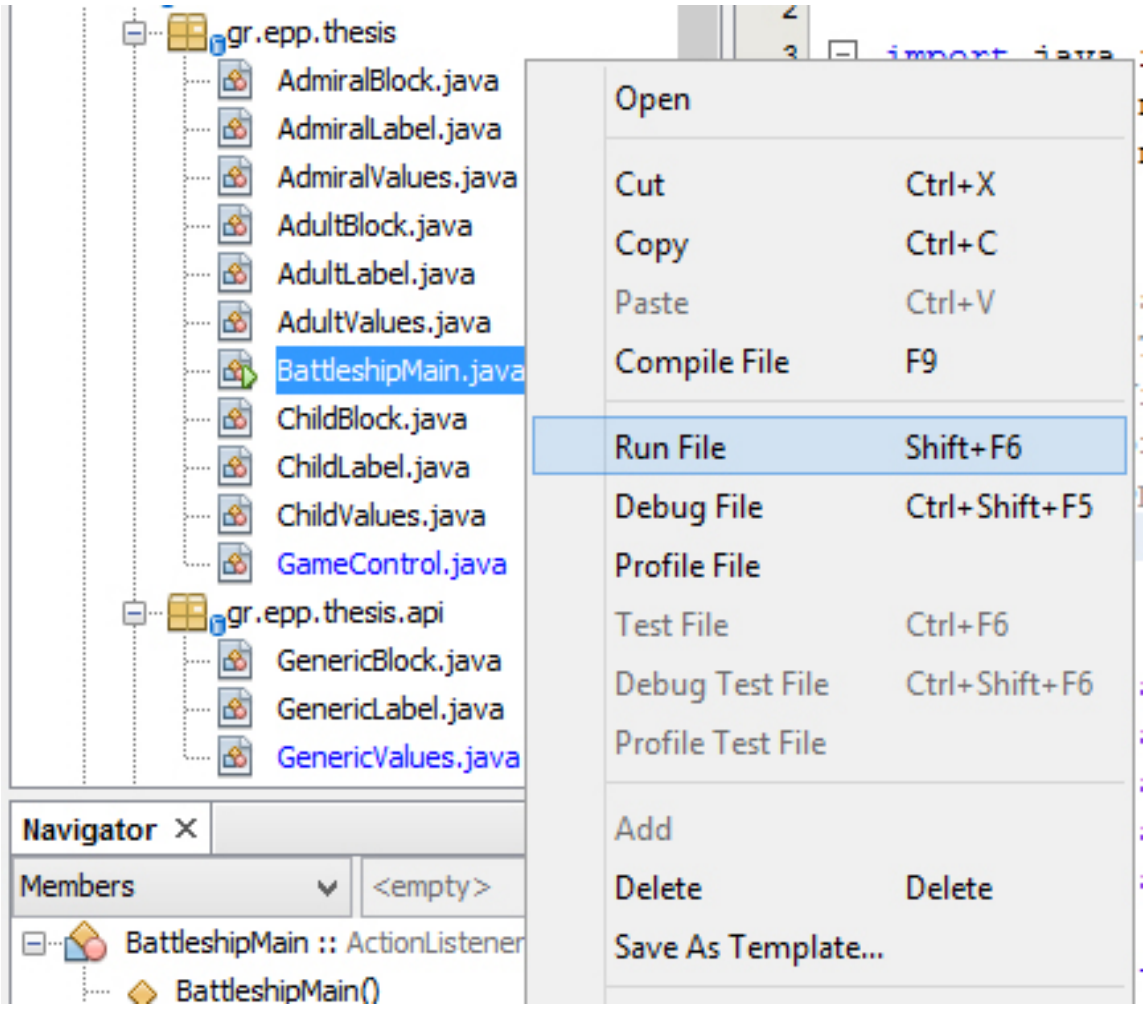
εικόνα 20 *- εκκίνηση server*

Αρχικά θα πρέπει να εκκινήσουμε τον server, που είναι η κλάση **Server.java** και βρίσκεται μέσα στο package **gr.epp.thesis.tcp**, κλάση την οποία θα εξηγήσουμε αργότερα όσο αφορά τη δομή της και τις μεθόδους τις. Ο **εξυπηρετητής** (server) εκκινείτε και μας ενημερώνει, στο output, πως ακούει (listening) στο **port 1501** και είναι έτοιμος να δεχτεί clients.



εικόνα 21 *- output της server.java*

Ύστερα, ανοίγοντας το πακέτο **gr.epp.thesis** βρίσκουμε την κλάση **BattleshipMain.java**, όπου και είναι η κλάση που πρέπει να τρέξει ο κάθε client αντίστοιχα τώρα, δηλαδή ο παίκτης. Στην περίπτωση μας εδώ, εφόσον εξετάζουμε ένα παράδειγμα, τρέχουμε από το ίδιο pc 2 φορές την ίδια κλάση, ούτως ώστε να έχουμε 2 συνδεδεμένους παίκτες και να συνεχίσουμε.



εικόνα 22 *- εκκίνηση client*

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

Στη συνέχεια μας εμφανίζεται ένα παράθυρο που μας δίνει την επιλογή να επιλέξουμε ανάμεσα σε είδος παίκτη. Εμείς, στην έκδοση αυτή του παιχνιδιού, έχουμε διαθέσιμους στη λίστα μας τρία είδη παικτών, **adult** (ενήλικας), **child** (παιδί), και μια προσθήκη έμπειρου και δύσκολου παίκτη, του **admiral** (ναύαρχος). Για την δοκιμή αυτή λοιπόν, αποφασίζουμε να επιλέξουμε τον πρώτο παίκτη για adult.

Ακολούθως, τρέχουμε ξανά την κλάση BattleshipMain.java και αυτή τη φορά επιλέγουμε για δεύτερο παίκτη, το child. Ακολουθούν εικόνες με τις επιλογές μας αναλυτικά, ώστε να έχουμε πιο απτά και κατανοητά παραδείγματα στη διάθεσή μας.

εικόνα 23 *- επιλογή παίκτη*

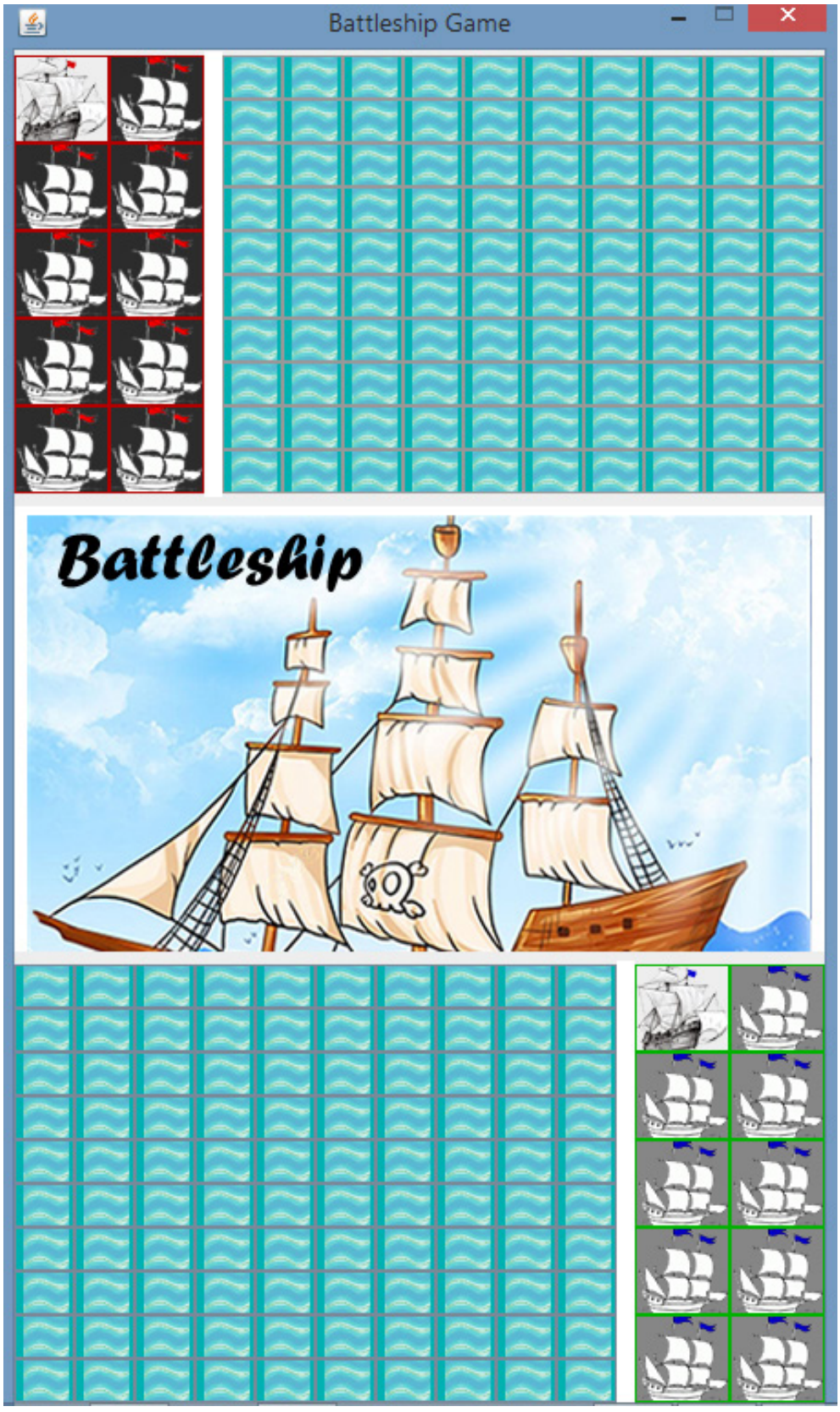
Παραθέτουμε σε εικόνες επίσης παρακάτω πως είναι τα περιβάλλοντα χρήσης του παιχνιδιού για τα δυο παραδείγματα παικτών. Από το περιβάλλον λοιπόν του κάθε παίκτη παρατηρούμε πως για το παιδί έχουμε επιλεγεί πιο ήπια χαρακτηριστικά, μια καρτουνίστικη διάθεση δηλαδή, ένα μοτίβο πιο πειρατικό, εικόνες πιο ζωγραφιστές, οτιδήποτε θα μπορούσε να φαντάζει πιο οικείο σε ένα παιδί που δοκιμάζει να μάθει και να παίξει το παιχνίδι. Για τον ενήλικα όμως το περιβάλλον είναι πιο ουδέτερο, πιο κοντά στο κλασσικό στυλ της Ναυμαχίας που βρίσκει κανείς αν ψάξει στο internet, για εκδόσεις του. Από τα πολεμικά πλοία μέχρι και τα γραφικά είναι όλα διαφορετικά για κάθε είδος παίκτης ούτως ώστε να δώσουμε έμφαση στο αντικείμενο του **πολυμορφισμού** που εκμεταλλεύεται η πτυχιακή μας εργασία. Ο παίκτης child έχει στη διάθεση του πλοία με κατάρτια, δηλαδή ιστιοφόρα, ενώ ο παίκτης adult έχει πλοία πιο σύγχρονα.

Επίσης, οι κανόνες έχουν εμφανώς αλλάξει, δηλαδή έχουμε έρθει πιο πολύ στα μέτρα του παίκτη child, ώστε να είναι πιο εύκολο να νικήσει το παιδί. Τα πλοία του παίκτη child καταλαμβάνουν ένα block μόνο στο πλέγμα (grid), και έτσι είναι πιο δύσκολο για τον αντίπαλο παίκτη, εν προκειμένη περίπτωση ο adult, να τα βρει και να τα βυθίσει.

Αντίστοιχα ο τελευταίος έχει την κανονική (default) έκδοση του παιχνιδιού, με τον κάθε τύπο πλοίων να πιάνει όσο χώρο πρέπει σύμφωνα με του κανόνες του παιχνιδιού, κανόνες τους οποίους μπορεί εύκολα κανείς να βρει στο internet αν ανατρέξει. Καταλαβαίνουμε στο σημείο αυτό, πως ο τύπος παίκτη adult έχει υπάρξει η βάση του project μας, μιας και αντιπροσωπεύει την **by default** έκδοση του παιχνιδιού, και πάνω σε αυτόν βασίστηκε το χτίσιμο και των υπολοίπων δυο παικτών μας.

εικόνα 24 *- περιβάλλον χρήσης παίκτη adult*

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389



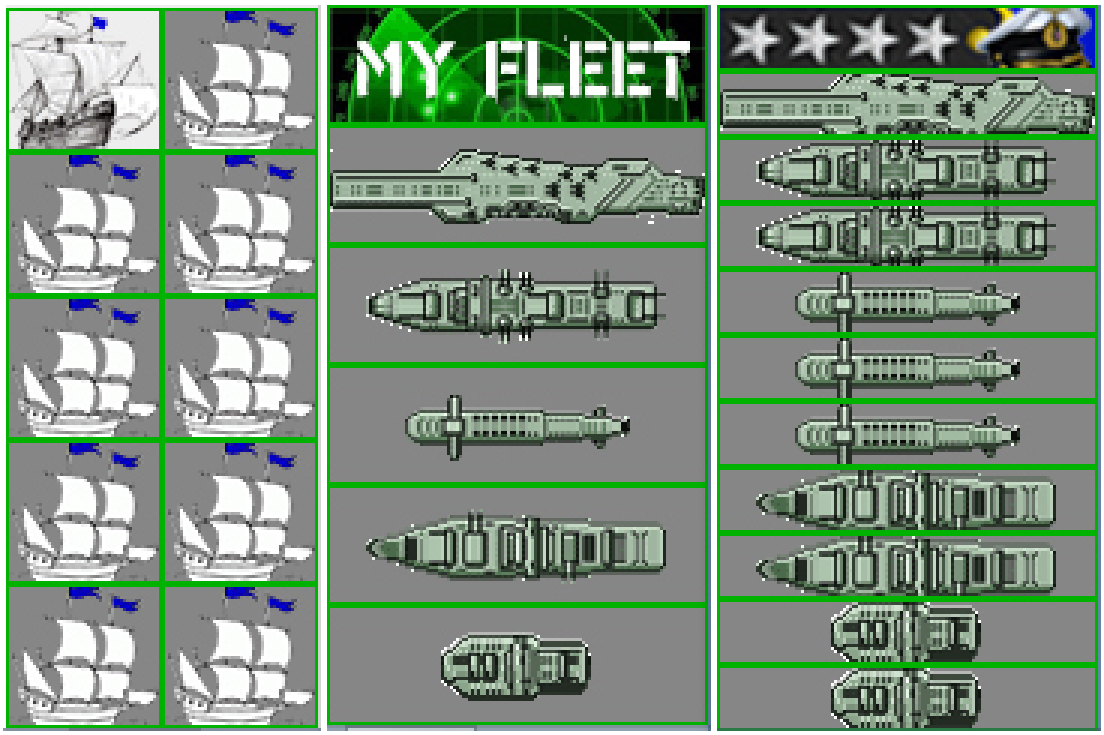
εικόνα 25 *- περιβάλλον χρήσης παίκτη child*

Αν και δεν το εξετάζουμε στην παρούσα περίπτωση θεωρούμε ευκαιρία να παρουσιάσουμε και το πιο εντυπωσιακό και προκλητικό περιβάλλον χρήσης από τα υπόλοιπα δυο περιβάλλοντα, του παίκτη **admiral**, που αντιπροσωπεύει στο project μας την επιλογή του expert παίκτη προφανώς.

εικόνα 26 *- περιβάλλον χρήσης παίκτη admiral*

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

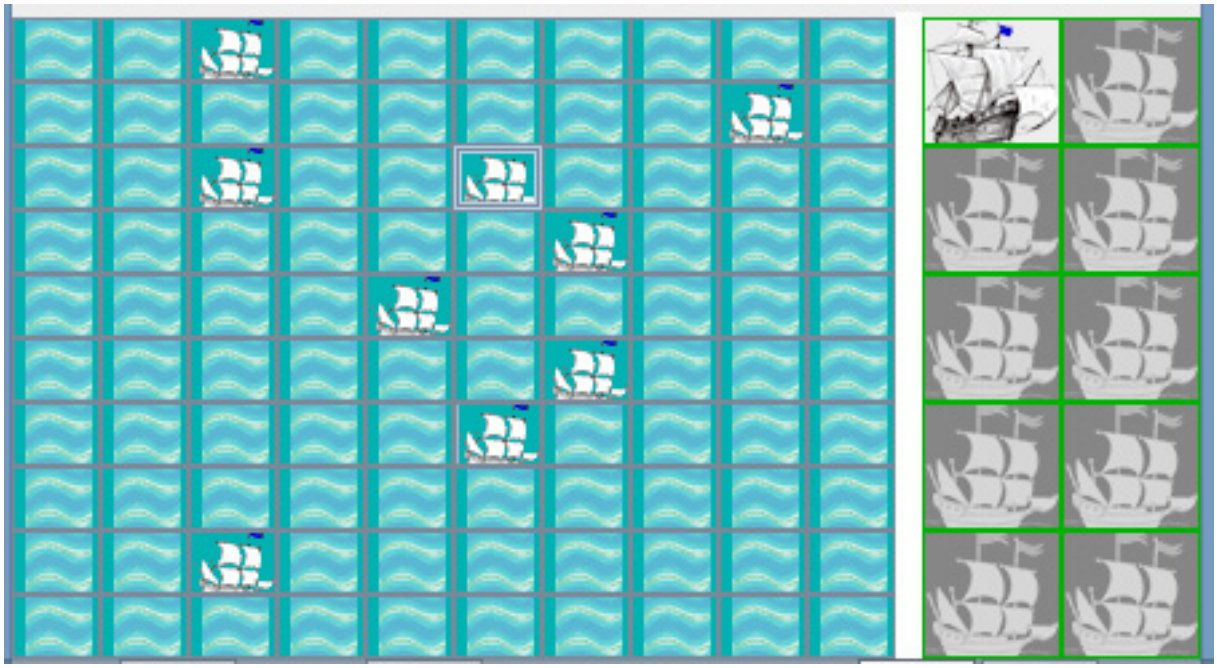
Από τις εικόνες που μόλις είδαμε, διακρίνουμε 2 λίστες πλοίων. Η μια, που βρίσκεται κάτω δεξιά, είναι η δική μας, τα δικά μας διαθέσιμα πολεμικά πλοία, και η πάνω αριστερά είναι τα συνολικά πλοία του αντιπάλου που στόχος μας είναι να τα καταστρέψουμε εκτελώντας βολές ώσπου να χτυπήσουμε κάθε block, κάθε πλοίου.



εικόνα 27 *- λίστες πολεμικών πλοίων ανα παίκτη*

Βρισκόμαστε λοιπόν αυτή τη στιγμή στη με δυο παίκτες έτοιμους να στήσουν τα πλοία τους και να προωθηθούν στην διαδικασία βολών, όπου είναι και ο κύριος όγκος του χρόνου που απασχολεί του παίκτες στο παιχνίδι. Στον υπολογιστή μας, όπως αναφέραμε, προκειμένου να εξηγήσουμε λεπτομερώς το παράδειγμα, οι παίκτες adult και child τρέχουν το project μαζί, δηλαδή στον ίδιο υπολογιστή, άρα και η αναμέτρηση έχει χαρακτήρα επεξηγηματικό.

Πριν ξεκινήσει η μάχη, υπάρχει η διαδικασία του στησίματος των πολεμικών πλοίων επάνω στο grid. Για τον παίκτη child 9 πλοία μεγέθους ένα block το καθένα όπου μπορεί να τα στήσει όπου επιθυμεί σε ένα πλέγμα διαστάσεων **10 * 10**, σύνολο δηλαδή 100 block θάλασσας, ή 100 block κενών θέσεων ας πούμε. Ένα παράδειγμα προετοιμασίας και στησίματος θα μπορούσε να είναι το παρακάτω.



εικόνα 28 *- παράδειγμα προετοιμασίας παίκτη child*

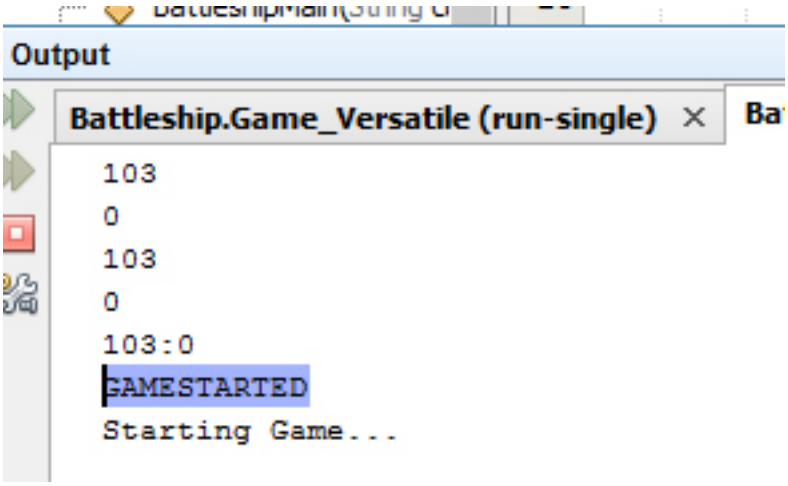
Εξαιρετικά και μόνο για τον παίκτη admiral αυξήσαμε το grid σε 15 * 15 κενές θέσεις, μιας και τα πλοία είναι τα διπλάσια, και γενικά αυτή η έκδοση του παιχνιδιού σίγουρα έχει περισσότερες απαιτήσεις. Να αναφέρουμε στο σημείο αυτό πως μόνο του ίδιου τύπου παίκτες μπορούν να αναμετρηθούν σε αυτό το επίπεδο, δηλαδή Admiral vs Admiral.

Αντίστοιχα, για τον παίκτη adult όπου διαθέτει 5 πολεμικά πλοία με διάφορα μεγέθη (5 blocks για το αεροπλανοφόρο, 4 blocks για το θωρηκτό, 3 blocks για το υποβρύχιο, 2 blocks για τη φρεγάτα, και 1 block για την τορπιλάκατο) ένα παράδειγμα στησίματος και προετοιμασίας, επάνω στο ίδιο grid (**10 * 10 κενές θέσεις**) θα μπορούσε να είναι αυτό.

εικόνα 29 *- παράδειγμα προετοιμασίας παίκτη adult*

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

Αυτόματα, εφόσον στήσουν και οι 2 παίκτες τα πλοία τους στο πλέγμα και αδειάσει η λίστα του στα δεξιά όπου είναι η λίστα με τα διαθέσιμα πολεμικά πλοία τους κατά την εκκίνηση του παιχνιδιού, το output μας ενημερώνει πως το παιχνίδι ξεκινά. Ο κώδικας έχει δημιουργηθεί έτσι ούτως ώστε το παιχνίδι να μην ξεκινά, και να μην δέχεται βολές στο grid του αντιπάλου αν και οι 2 παίκτες δεν έχουν ολοκληρώσει την πρώτη φάση.



εικόνα 30 *- output εκκίνησης παιχνιδιού*

Τηρείται σειρά προτεραιότητας, και έχει προβλεφθεί από τον κώδικα, δηλαδή πρώτος μπορεί να εκτελέσει βολή ο παίκτης child, και όχι ο adult μιας και συνδέθηκε πρώτος στον server. Το αντίπαλο grid λοιπόν για τον adult, στο παράθυρό του, είναι απενεργοποιημένο.

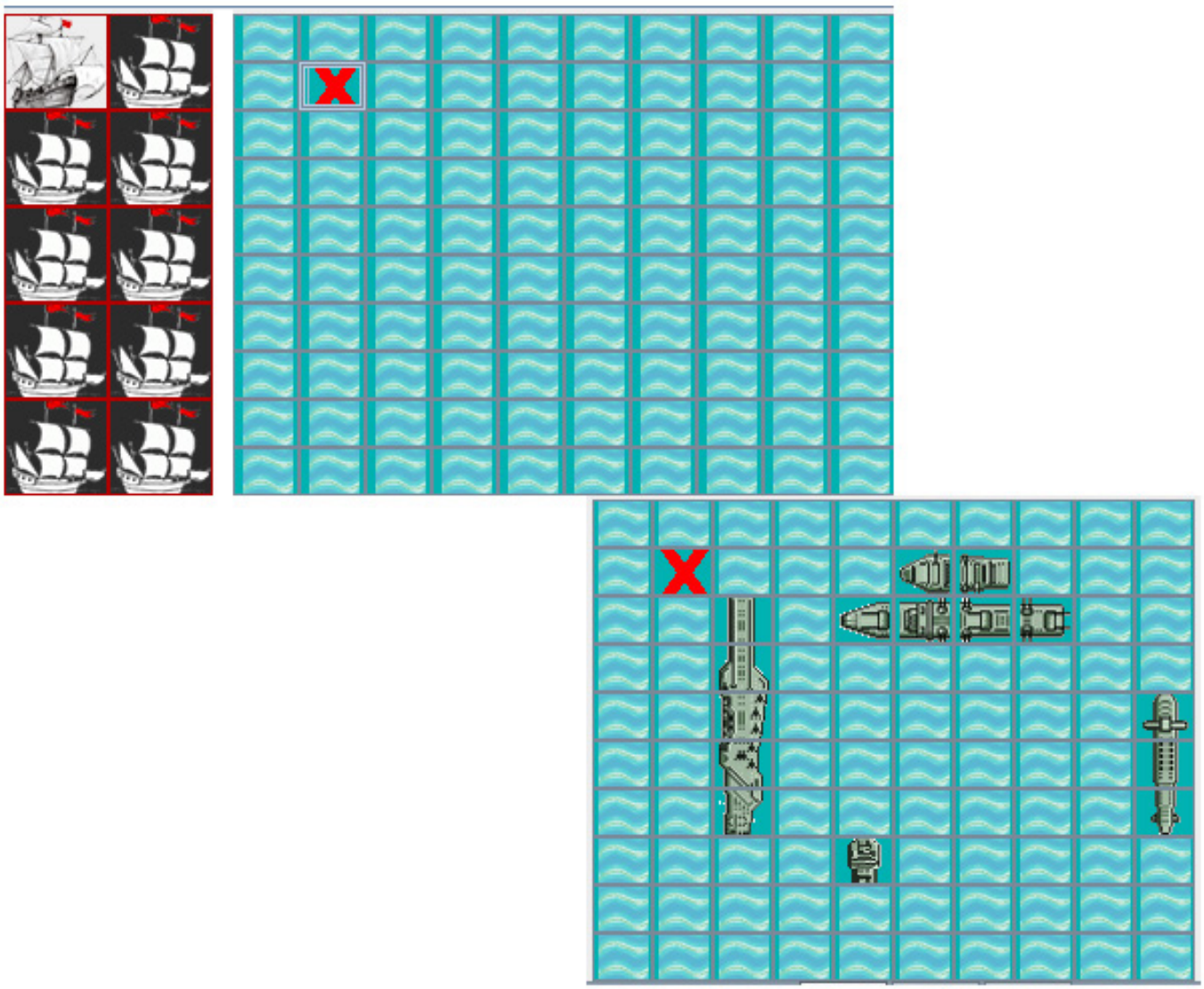
Επιχειρεί όπως βλέπουμε παρακάτω την πρώτη βολή του ο παίκτης child, η οποία είναι αποτυχημένη. Από το output του NetBeans εκτυπώνεται και σχετικό μήνυμα κάθε φορά, που μας ενημερώνει για την ευστοχίας μας (accuracy), και για το αν ήταν επιτυχημένη η βολή στο συγκεκριμένο block. Στην πρώτη βολή που εκτέλεσε ο παίκτης child στη θέση 11, η βολή είχε αποτέλεσμα missed.

Στη συνέχεια έρχεται η σειρά του παίκτη adult, ο οποίος και αυτός με τη σειρά του επιχειρεί βολή στη θέση 81, και το αποτέλεσμα κρίνεται missed (αποτυχημένο) επίσης. Για τον παίκτη adult πάλι εκτυπώνεται σχετικό μήνυμα για την ευστοχία του, στο output του IDE μας.

Ακολουθούν εικόνες από τις περιπτώσεις που εξηγούμε, ώστε να έχουμε καλύτερη κατανόηση της δομής του παιχνιδιού αλλά και του τρόπου με τον οποίο εκτελούνται οι βολές.

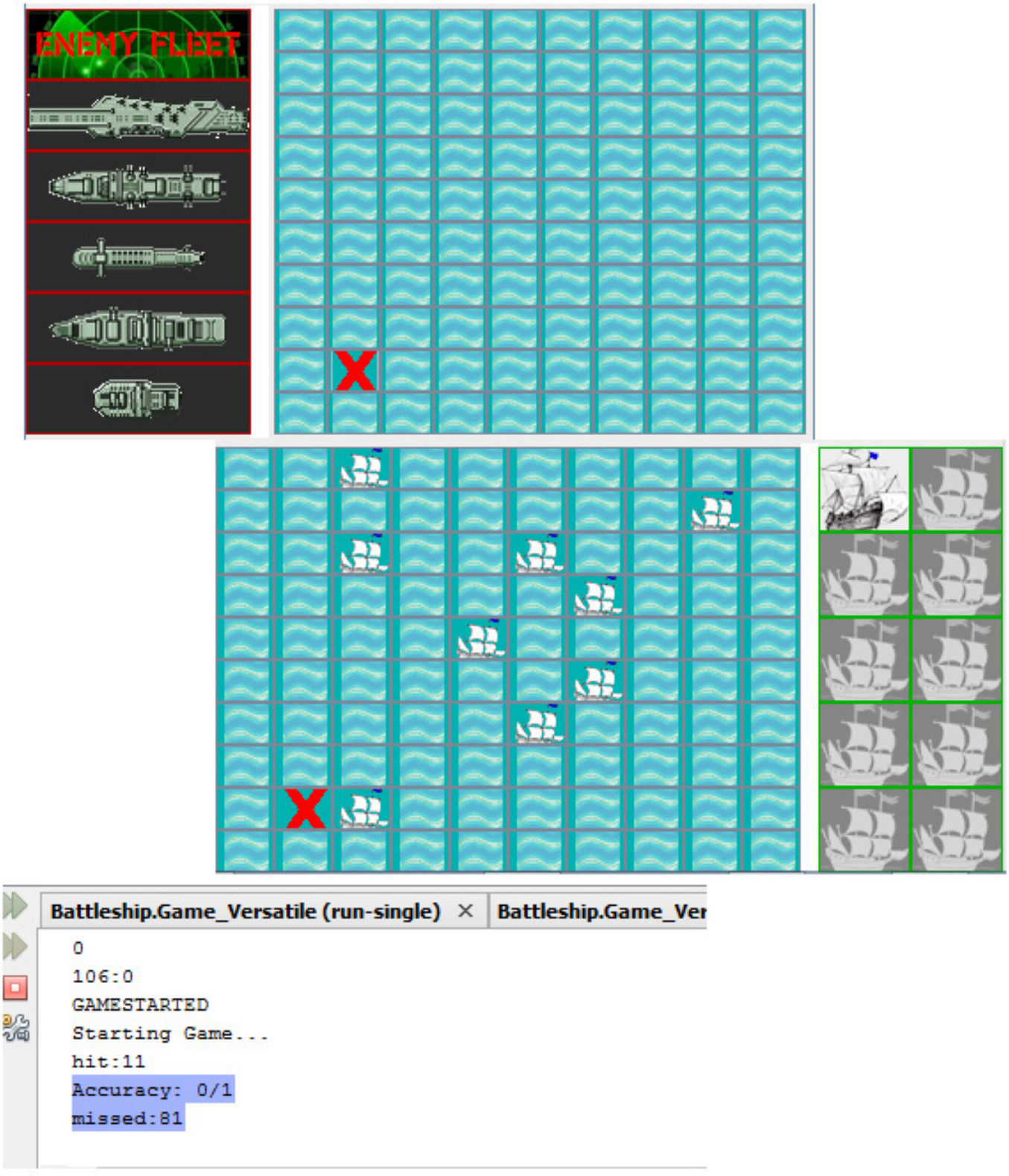
Ευκαιρία να αναφέρουμε πως ο κέρσορας καθώς αιωρείτε πάνω από το πλέγμα του αντιπάλου (αντίπαλο grid) αλλάζει σε γραφικό ξίφους, επάνω από τα blocks όπου κρύβονται τα εχθρικά πλοία, στην περίπτωση παίκτη τύπου child.

εικόνα 31 *- γραφικό ξίφους*



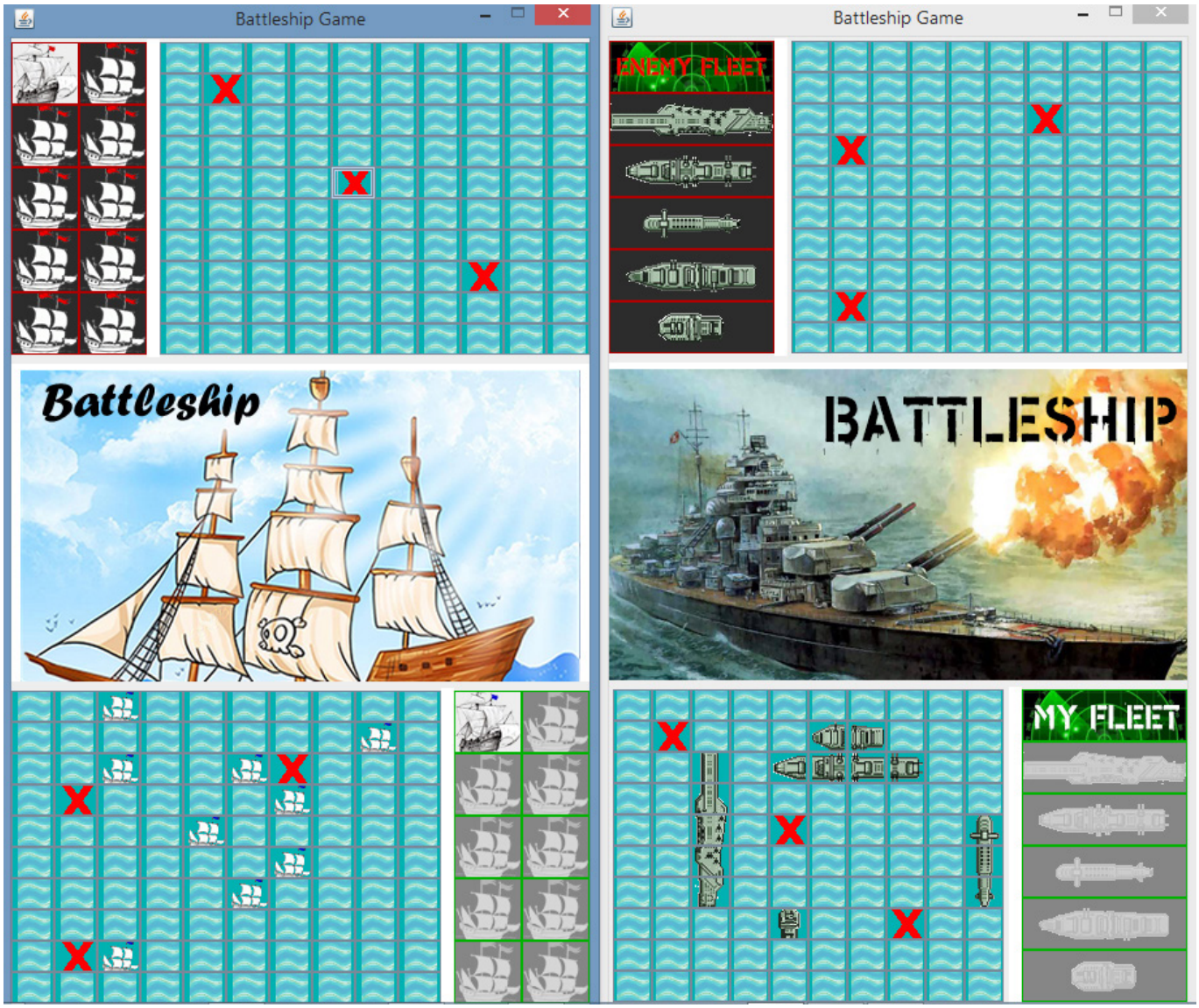
εικόνα 32 *- η πρώτη αποτυχημένη βολή*

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389



εικόνα 33 *- η δεύτερη αποτυχημένη βολή*

Λίγα χτυπήματα αργότερα εκατέρωθεν, οι αντίπαλοι παραμένουν με 0 επιτυχημένες βολές. Όσο καλύπτουν με αποτυχημένες βολές το grid, τόσο πιο κοντά θα είναι στην επιτυχία η επόμενη βολή τους, και στην αποκάλυψη των θέσεων του αντιπάλου τους. Βέβαια ο παίκτης child εδώ έχει καθαρό πλεονέκτημα, μιας και αν πετύχει ένα block, στις επόμενες βολές του έχει πολύ μεγάλη πιθανότητα να βυθίσει όλο το πλοίο γρήγορα διότι τα blocks που έχουν πάνω τους το ίδιο πλοίο είναι γειτονικά, οριζόντια ή κάθετα



εικόνα 34 *- συνεχόμενες βολές*

Ακόμα μερικές τυχαίες βολές αργότερα παρατηρούμε πως έχει δημιουργηθεί μια μεγάλων διαστάσεων ναυμαχία!! Συγκεκριμένα, ο παίκτης child έχει βυθίσει το αεροπλανοφόρο (με 5 blocks μέγεθος είναι πολύ εύκολο κανείς να το ανακαλύψει), και έχει πλήξει με 2 βολές το υποβρύχιο επίσης όπου μένει μόνον μια βολή ώστε να το αποτελειώσει. Ο παίκτης adult έχει με τη σειρά του καταφέρει να βυθίσει 4 πλοία του παίκτη child. Θα λέγαμε πως ο παίκτης adult είναι πιο επιτυχημένος στις βολές του καθώς είναι πολύ δύσκολο να βρει πλοία μεγέθους ενός block σε ένα grid 10 * 10. Παρακάτω η εικόνα με τις βολές, επιτυχημένες και μη.

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

εικόνα 35 *- ναυμαχία σε εξέλιξη*

Συνεχίζοντας με την ίδια δυναμική το παιχνίδι, οι αντίπαλοι ανταλλάσουν βολές δια μέσου του TCP, όπου δεν κάνει κάτι άλλο παρά να στέλνει πακέτα με **strings** από τον έναν client στον άλλον, αφήνοντας μετά τον κώδικα να εξετάσει αν στο block που αντιστοιχεί του νούμερο που μετέφερε το string είναι ελεύθερο (block θάλασσας) ή κατειλημμένο (block πολεμικού πλοίου). Συνεχίζεται έτσι λοιπόν ώσπου κάποιος από τους δυο να βυθίσει πρώτο τα πλοία του αντιπάλου.

Θα μπορούσε ύστερα από λίγα λεπτά το θαλάσσιο πεδίο της ναυμαχίας να είναι όπως δείχνει η εικόνα παρακάτω, και να έχει κερδίσει ο παίκτης child για παράδειγμα.

εικόνα 36 *- νίκη του παίκτη child*

Ανάλογα με την έκβαση της μάχης, τέλος, στον κάθε παίκτη που διαγωνίστηκε, θα εμφανιστεί ένα από τα 2 παρακάτω μηνύματα σε εικόνα. Και κάπως έτσι φτάνει στο τέλος της η Ναυμαχία.

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

εικόνα 37 *- μήνυμα σε περίπτωση νίκης*

εικόνα 38 *- μήνυμα σε περίπτωση ήτας*

Αν και μέσα στον κώδικα με τα σχόλια που χρησιμοποιούμε στις κλάσεις, αλλά και για κάθε μέθοδο και μεταβλητή που χρησιμοποιείτε ξεχωριστά, δηλώνουμε τη χρησιμότητά της σχετικά, θεωρούμε αναγκαίο να σταθούμε αναλυτικότερα στην επεξήγηση του περίπλοκου αυτού κώδικα.

4.2. Η κλάση GameControl και οι μεθόδοι της:

Η κλάση GameControl στην ουσία ελέγχει και διαχειρίζεται όλο το παιχνίδι. Από τη στιγμή που σχεδιάζεις πως θα στήσεις τα πλοία σου, ως και το τέλος του παιχνιδιού, όλος ο κώδικας που θα χρειαστεί να εκτελεστεί για να αλληλεπιδράσουν τα grid blocks μεταξύ τους, βρίσκεται εκεί.

Αρχικά, η κλάση θα πρέπει να υλοποιεί το **MouseListener interface**, καθότι οποιοδήποτε στοιχείο του grid όταν εγκαθιστάτε επάνω σε αυτό υποχρεωτικά πρέπει να υλοποιεί την GameControl. Επίσης υποχρεωτικά η κλάση πρέπει να υλοποιεί το **Runnable interface**, και κατ' επέκταση θα κάνει Override την **run** μέθοδο όπου μας είναι απαραίτητη ώστε να λαμβάνουμε και να αποστέλλουμε δεδομένα όσο αφορά τη διαδικασία της μάχης με τον αντίπαλο. Με λίγο λόγια τα hits ή τα miss των 2 παικτών.

Ας περάσουμε όμως παρακάτω στην επεξήγηση του κώδικα της συγκεκριμένης κλάσης.

Constructor() :

Όπως παρατηρεί κανείς στην BattleshipMain κλάση του project, όπου και βρίσκεται το entry point του, όταν γίνεται instantiate η GameControl [gameControl = new GameControl(playerValues);] της ορίζεται ένα όρισμα όπου και περνά στον constructor της.

Το όρισμα αυτό όπως είναι εμφανές και από το όνομα που έχει επιλεχθεί εμπεριέχει όλες τις μεταβλητές που έχουν προκαθορισμένα οριστεί (by default) από εμάς, για το συγκεκριμένο τύπο παίκτη. Θέλω να πω.. Ένας adult παίκτης, από εμάς έχει επιλεχθεί να παίξει την κανονική έκδοση (standard version) της ναυμαχίας, χωρίς προσθήκες ή παραλλαγές. Ένας παίκτης τύπου child έχει οριστεί να δοκιμάσει μια διαφορετική έκδοση του παιχνιδιού, δηλαδή πιο light και νηπιακή, αν θέλετε.. Όπως είναι φυσικό, άλλες εικόνες και άλλα γραφικά έχουν επιλεχθεί για το παιδί και άλλες για τον ενήλικα. Μια πιο εύκολη εκδοχή του παιχνιδιού, επίσης, έχει οριστεί όταν παίζει ένα παιδί πίσω από τον υπολογιστή με μόλις 9 πλοία στο grid όπου το καθένα πιάνει μόνον μια θέση. Για τον λόγο αυτό χρειαζόμασταν διαφορετικές μεταβλητές για τον καθένα, ανάλογα με το είδος του παίκτη, άρα και μια κλάση με τις μεταβλητές κάθε είδους παίκτη δημιουργήθηκε. Αυτή η συγκεκριμένη κλάση κάθε φορά απαραίτητα πρέπει να "περνά" σαν όρισμα στην Game Control όπου και λαμβάνει χώρα στην ουσία όλη η ραχοκοκαλιά του παιχνιδιού.

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

```
public GameControl(GenericValues playerValues, Socket clientScoket,
        String currentPlayerType) {
    this.playerValues = playerValues;                //Current player values.
    this.gridRows = playerValues.getGridRows();
    this.gridColumns = playerValues.getGridColumns();
    this.shipBlocksTotalNumber = playerValues.getShipBlocksTotalNumber();
    this.seaColor = playerValues.getSeaColor();
    this.waterIcon = playerValues.getWaterIcon();
    this.successfulShotIcon = playerValues.getSuccessfulShotIcon();
    this.missedShotIcon = playerValues.getMissedShotIcon();
    this.warshipBlocksList = new ArrayList<>();
    this.hittenBlocks = new ArrayList<>();
    this.clientSocket = clientScoket;
    this.currentPlayerType = currentPlayerType;
    try {
        in = new BufferedReader(new InputStreamReader(clientSocket.
                getInputStream()));
        out = new PrintWriter(clientSocket.getOutputStream(), true);
    } catch (IOException ex) {
        Logger.getLogger(GameControl.class.getName()).
                log(Level.SEVERE, null, ex);
    }
    out.println("notify:" + currentPlayerType);
}
```

εικόνα 39 *- ο constructor της GameControl*

Μέθοδος setLateValues() :

Αργότερα χρειαζόταν μια μέθοδος για να περάσουμε κάποιες παραμέτρους στην Game Control, αφού όμως έχουν γίνει **instantiate** οι κλάσεις οι αναγκαίες ανά περίπτωση. Κάθε παίκτης που επιλέγεται να ξεκινήσει, ικανοποιεί διαφορετικές μεταβλητές και ύφος στο παιχνίδι του. Το αποτέλεσμα μπορεί να επιτυγχάνεται με τον ίδιο κώδικα για κάθε τύπο παίκτη, αλλά υπάρχουν πράγματα που αλλάζουν, π.χ.: γραφικά, χρώματα, υπολογισμός πόντων, και άλλα. Οι μεταβλητές αυτές που είναι μοναδικές ανα παίκτη, και οι κλάσεις τους, καλούνται λοιπόν ανάλογα τον τύπο παίκτη που ξεκινά το παιχνίδι.

Νωρίτερα ήταν αδύνατο να τα περάσουμε με ορίσματα κι αυτά στον constructor, γιατί δε γνωρίζαμε με ποιόν παίκτη έχουμε να ασχοληθούμε, άρα και με μια μέθοδο ύστερα τα περνάμε αμέσως μετά. Η μέθοδος αυτή είναι η setLateValues().

Μέθοδος activateBoard() :

Αυτή η μέθοδος ενεργοποιεί και απενεργοποιεί όλα τα στοιχεία του grid, όταν αυτό χρειάζεται.

Όπως μπορεί να καταλάβει κανείς, άμεσα χρησιμοποιείτε η μέθοδος όταν ένας παίκτης έχει εκτελέσει τη βολή του στον αντίπαλο, και περιμένει απάντηση. Το grid που χρησιμοποιεί για να επιλέξει τις βολές του θα πρέπει να είναι απενεργοποιημένο τη στιγμή εκείνη. Επίσης, η μέθοδος χρησιμοποιεί διαφορετικές εικόνες όταν είναι απενεργοποιημένο το κάθε block, αλλά και αφαιρεί από το κάθ' ένα τον Listener.

Μέθοδος getBlockPosition() :

Κάθε στοιχείο του grid (block) πρέπει να έχει μια συγκεκριμένη θέση επάνω σε αυτό. Κατά την εκτέλεση του κώδικα χρειάζεται να έχουμε άμεσα, όταν χρειαστεί, πληροφορίες σχετικά με τη θέση του επιλεγμένου κάθε φορά block επάνω στο grid.

Αυτό που συμβαίνει λοιπόν, σε αυτήν τη μέθοδο, είναι πως έχουμε ένα προσωρινό πίνακα, τον coords[] με 3 θέσεις. Η πρώτη θέση αντιστοιχεί στην συντεταγμένη στο οριζόντιο επίπεδο (**τετμημένη**), και η δεύτερη θέση του πίνακα αντιστοιχεί στο κάθετο επίπεδο (**τεταγμένη**). Δηλαδή με δυο μεταβλητές μπορούμε να γνωρίζουμε κάθε φορά με ακρίβεια τις συντεταγμένες του block επάνω στο grid. Για να βρούμε την τετμημένη, απλώς διαιρούμε τον αριθμό της θέσης του block με τον συνολικό αριθμό των γραμμών του grid, ενώ για την τεταγμένη πρέπει να βρούμε το modulo (υπόλοιπο από μια απόλυτη διαίρεση) της θέσης του block με τον συνολικό αριθμό των στηλών του grid.

Η τρίτη μεταβλητή του πίνακα μας απλώς αποθηκεύει την θέση του block, αν τα μετρήσει κανείς από αριστερά προς τα δεξιά, οριζόντια. Μας δείχνει σε ποια σειρά δηλαδή το καταχώρησε αρχικά το Layout που έχουμε χρησιμοποιήσει για το grid.

```
public void getBlockPosition(GenericBlock block) {
    JPanel currentParent = (JPanel) block.getParent();
    for (int i = 0; i < currentParent.getComponentCount(); i++) {
        if (currentParent.getComponent(i) == block) {
            coords[0] = i / gridRows; // rows, columns?
            coords[1] = i % gridColumns; // rows, columns?
            coords[2] = i; // BlockPosition
            break;
        }
    }
}
```

εικόνα 40 *- καταχώρηση των συντεταγμένων*

Μέθοδος warshipBlockOnGrid() :

Η μέθοδος αυτή μας βοηθά να διαχειριστούμε κατάλληλα τα γραφικά του παιχνιδιού. Δηλαδή τις εικόνες που χρησιμοποιούνται για κάθε τύπο πλοίου, και κατ' επέκταση για κάθε τύπο παίκτη. Στα στοιχεία του grid που δεν έχουν block πολεμικού πλοίου εισάγουμε μια εικόνα στα χρώματα της θάλασσας.

Η μέθοδος καλεί με τη σειρά της μια άλλη μέθοδο από κλάση που βρίσκεται σε διαφορετικό package. Αναλυτικότερα, καλείτε η **getGridPieces()** από την κλάση **GenericValues** του πακέτου **gr.epp.thesis.api**, όπου είναι μια μέθοδος που επιστρέφει (κάνει **return**) μια μεταβλητή ImageIcon, δηλαδή μια μικρή εικόνα που τοποθετείτε στο επιλεγμένο block κάθε φορά. Η μέθοδος αυτή για να συνεργαστεί με επιτυχία και να επιστρέψει αυτό που μας χρειάζεται πρέπει να τροφοδοτηθεί με 3 ορίσματα, τον συνολικό αριθμό των block που διαθέτει ένα πλοίο (π.χ. το αεροπλανοφόρο έχει 5 blocks, η φρεγάτα 2 blocks), το συγκεκριμένο block που έχει σειρά να τοποθετηθεί τη στιγμή εκείνη (δηλαδή αν έχει επιλεχθεί το αεροπλανοφόρο, η μέθοδος αυτή θα κληθεί 5 φορές!), και η κατεύθυνση που το τοποθετούμε (orientation – 3: οριζόντια, 6: κάθετα), καθώς και μια boolean, που μας ορίζει αν το συγκεκριμένο ImageIcon αντιπροσωπεύει χτυπημένο block από κάποιο πλοίο.

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

```
public void warshipBlockOnGrid(GenericBlock warshipBlock, int currentBlock) {
    this.l = 0;
    warshipBlock.setIcon(playerValues.getGridPieces(shipBlocksNumber,
            currentBlock, orientation, false));
    System.out.println("" + warshipBlock.getName());
    System.out.println(warshipBlock.getIndex());

    warshipBlock.setBackground(seaColor);
    warshipBlock.setWarshipBlockOnGrid(true);
    currentWarship.setEnabled(false);
    warshipBlocksList.add(currentWarship);
    for (int i = 0; i < alliesBoard.getComponentCount(); i++) {
        if (alliesBoard.getComponent(i) == warshipBlock) {
            this.warshipBlocksHold[l] = i;
            l++;
        }
    }
}
```

εικόνα 41 *- μέθοδος warshipBlockOnGrid()*

Στον πίνακα παρακάτω παρουσιάζουμε τυχαία παραδείγματα του τι μπορεί να συναντήσει κάποιος παίκτης επάνω στο grid.

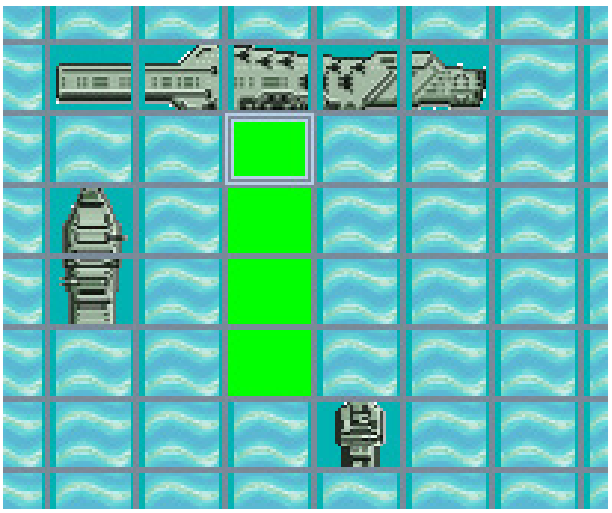
	Το πρώτο block, από τα 3 που διαθέτει το υποβρύχιο, σε θέση οριζόντια.
	Το δεύτερο block, απο τα 5 που διαθέτει το αεροπλανοφόρο, σε θέση κάθετη, και χτυπημένο.
	Το ImageIcon που χρησιμοποιούμε για τη θάλασσα.
	Το block ενός πλοίου σε επιλογή παίκτη για παιδί. (Child)

<u>Μέθοδος checkCollision() :</u>

Ο περίπλοκος αυτός κώδικας, μας βοηθά να ελέγξουμε κάθε φορά που προσπαθούμε να εγκαταστήσουμε ένα πλοίο στο grid, αν κάποιο από τα block που δεσμεύουμε είναι ήδη μέρος πολεμικού πλοίου που έχουμε εγκαταστήσει νωρίτερα. Όπως είναι λογικό, δεν μπορούμε να χρησιμοποιήσουμε το ίδιο block για 2 πλοία.

Η επιλογή μας κάθε φορά, πριν αποφασίσουμε και στήσουμε το πλοίο, μας αναπαριστάνεται με πράσινο χρώμα και "αιωρείτε" (**hover**) πάνω από το grid, ανάλογα με την πορεία του κέρσορά μας. Όταν η κέρσοράς μας περάσει πάνω από ένα ήδη εγκατεστημένο πλοίο το πράσινο χρώμα εξαφανίζεται και η επιλογή μας προσωρινά δεσμεύεται, διότι η μέθοδος έχει ανιχνεύσει σύγκρουση "collision", και δε θα μας

επιτρέψει να τοποθετήσουμε το πλοίο σε εκείνες τις θέσεις. Τέλος, να αναφέρουμε ότι η μέθοδος μπορεί να ελέγχει για ελεύθερο χώρο στην ουσία οριζόντια και κάθετα.



εικόνα 42 *- hovering επάνω στο grid*

Σε αυτό το παράδειγμα όπου είμαι στην διαδικασία τοποθέτησης των πλοίων μου επάνω στο grid, δηλαδή λίγο πριν ξεκινήσει η αναμέτρηση, έχω επιλέξει το θωρηκτό (4 blocks γι' αυτό το πλοίο) και ορθά έχω ετοιμαστεί να το στήσω διότι έχω τον απαιτούμενο χώρο.

Αν όμως επιχειρήσω να το τοποθετήσω μια θέση πιο πάνω ο κώδικας θα ανιχνεύσει την σύγκρουση (collision) και δε θα με αφήσει να ολοκληρώσω.

Μέθοδος battleFormations() :

Μια γενικευμένων εργασιών μέθοδος, όπου αποτελείτε από πολλούς ελέγχους και μεταβλητές. Στην ουσία διαχειρίζεται όλες τις αλληλεπιδράσεις με το grid, αλλά και καλεί τις ανώτερο μεθόδους, έμμεσα, που περιγράψαμε, προκειμένου να εκτελέσει σωστά τη δουλειά της. Αν είμαστε στην επιλογή του hovering (αιώρηση), δηλαδή ένα βήμα πριν δεσμεύσουμε χώρο για κάποιο πολεμικό πλοίο προσωρινά αφαιρεί την εικόνα της θάλασσας και προσθέτει πράσινο χρώμα στο block δίνοντας μας την αίσθηση πως αναζητούμε "από ψηλά" τον κατάλληλο ελεύθερο χώρο. Αν πάλι περάσουμε πάνω από το block αυτό και συνεχίσουμε, δηλαδή τελείωσε η αιώρηση πάνω από το συγκεκριμένο block, επαναφέρει την ίδια κατάσταση πάλι στο τελευταίο (χρώμα της θάλασσας κλπ).

Αν δεσμεύσουμε το block όμως φροντίζει ώστε να εγκατασταθεί το πλοίο εκεί που επέλεξε ο παίκτης, και να αναπαρασταθεί γραφικά.

Λογικό είναι να περιμένουμε να εκτελεστεί η μέθοδος τουλάχιστον τόσες φορές όσες και το σύνολο των πολεμικών πλοίων που έχει ο παίκτης στη λίστα του. Επίσης, και αυτή η μέθοδος διαχειρίζεται κατάλληλα τον κώδικά της ώστε να μπορεί ο παίκτης να ελέγξει (**hover**) ή να εγκαταστήσει (**place**) το πλοίο του στο grid οριζόντια και κάθετα, και αυτό γίνεται με την μεταβλητή **orientation**.

Μέθοδος initiateGame() :

Η απλή διεργασία που εκτελείτε από τη μέθοδο αυτή είναι απλώς μια boolean μεταβλητή που αλλάζει σε **true**, και έτσι μπορεί το παιχνίδι να περάσει από τη φάση της στρατηγική τοποθέτησης των πολεμικών πλοίων, στη φάση των βολών προς τον αντίπαλο και εκατέρωθεν. Ενημερώνεται πρώτη, αμέσως μετά την ολοκλήρωση της τοποθέτησης των πλοίων. Ύστερα, δεν μπορεί κανείς να ξαναστήσει τα πλοία του διαφορετικά, καθώς η ναυμαχία έχει ξεκινήσει.

Μέθοδος run() :

Μια μέθοδος που περιέχει όλον τον απαραίτητο κώδικα όπου μας χρειάζεται ούτος ώστε να ανταλλάσσουμε δια μέσου του **server**, που ήδη τρέχει, strings όπου εμπεριέχουν

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

μέσα τους τις συντεταγμένες πάνω στις οποίες εκτελέσαμε βολή. Αντίστροφα, απαντά η ίδια μέθοδος με το αποτέλεσμα. Hit ή Miss!

Αρχικά καταχωρούμε σε μια μεταβλητή τύπου String το εισερχόμενο μήνυμα, και μετά το "διαιρούμε" στα 2, έχοντας το σημείο στίξης "**:**" ως το διαχωριστικό. Δηλαδή η εντολή **split** θα χωρίσει στη μέση το μήνυμα ακριβώς εκεί όπου θα συναντήσει το "**:**", και μετά θα καταχωρηθούν τα 2 String σε ένα πίνακα 2 θέσεων τύπου String επίσης.

```
@Override
public void run() {
    try {
          in = new BufferedReader(new InputStreamReader(clientSocket.getInpu
          out = new PrintWriter(clientSocket.getOutputStream(), true);
        while (true) {
            String incomingMessage = in.readLine();
            String[] splitMessage = incomingMessage.split(":");
            int hitBlock = Integer.parseInt(splitMessage[1]);
```

εικόνα 43 *- διαχωρισμός εισερχόμενου μηνύματος*

Όπως παρατηρούμε τον κώδικα βλέπουμε πως η πρώτη θέση του πίνακα splitMessage μπορεί να περιέχει 2 περιπτώσεις μηνύματος, "**hit**" ή **οτιδήποτε άλλο**. Αν το μήνυμα είναι hit, σημαίνει πως ο αντίπαλος παίκτης έχει επιχειρήσει βολή, σε κάποιο block όπου ορίζει η δεύτερη θέση του πίνακα. Η μεταβλητή hitBlock περιέχει έναν αριθμό, από μετατροπή του αρχικού μηνύματος που ήταν σε μορφή String, όπου αντιπροσωπεύει μια θέση επάνω στο grid. Δηλαδή, το μήνυμα που θα μπορούσε να επεξεργάζεται στην παρούσα περίπτωση είναι: "hit:32" για παράδειγμα, και μας δηλώνει ότι ο αντίπαλος παίκτης επιχείρησε βολή στη θέση 32. Στη συνέχεια εξετάζεται το συγκεκριμένο block, και αν έχει πάνω του θέση πλοίου (block κάποιου πλοίου), τότε η βολή είναι επιτυχημένη για τον αντίπαλο και τυπώνεται μήνυμα success, καθώς και η θέση. Αν η βολή ήταν σε block κενό δηλαδή στη θάλασσα, πάλι τυπώνεται μήνυμα (missed), και η θέση.

Συναντούμε εδώ την μέθοδο checkDestruction(), μια πολύ σημαντική και περίπλοκη μέθοδο. Θα την εξετάσουμε αργότερα όμως.

```
            if (splitMessage[0].equals("hit")) {
                GenericBlock playerHitBlock = (GenericBlock) alliesBoard.
                        getComponent(hitBlock);
                if (playerHitBlock.isWarshipBlockOnGrid()) {
                    out.println("success:" + hitBlock);
                    System.out.println(playerHitBlock.getWarshipName());

                      playerHitBlock.setIcon("Sink"+playerHitBlock.getWarshi
                    playerHitBlock.setIcon(successfulShotIcon);
                      checkDestruction(playerHitBlock, hitBlock);
                } else {
                    out.println("missed:" + hitBlock);
                    playerHitBlock.setIcon(missedShotIcon);
                }

                  out.println(refreshObservers:);
                locked = false;
            } else {
```

εικόνα 44 *- αποτέλεσμα βολής*

Στην περίπτωση που η πρώτη θέση του πίνακα splitMessage δεν είναι "hit" περιμένουμε οποιοδήποτε άλλο μήνυμα. Αυτό το οποιοδήποτε μήνυμα το διαχειριζόμαστε παρακάτω.

Πρέπει να απεικονίσουμε και στην δική μας οθόνη αν η βολή υπήρξε επιτυχής ή όχι. Κάθε παίκτης έχει το δικό του grid όπου βρίσκεται ο στόλος του, αλλά έχει και τοποθετημένο από πάνω απο το δικό του το grid όπου είναι κρυμμένα τα πλοία του αντιπάλου, άρα και θα πρέπει να αναγνωρίζει τις βολές του. Το μήνυμα λοιπόν που περιμένουμε θα είναι "**success**" ή "missed" συν τον αριθμό που υποδηλώνει τη θέση στο grid. (π.χ.: success:32 ή missed:11). Αντίστοιχα θα χρησιμοποιηθούν τα γραφικά στοιχεία με το αποτέλεσμα σε κάθε block, όπου για κάθε παίκτη είναι διαφορετικά.

Τυπώνεται επίσης ένα μήνυμα "Accuracy", όπου μας υποδηλώνει την ακρίβεια των βολών μας.

```
} else {
    GenericBlock enemyHitBlock = (GenericBlock) enemyBoard.
            getComponent(hitBlock);
    if (splitMessage[0].equals("missed")) {
        enemyHitBlock.setIcon(missedShotIcon);
    } else {
        enemyHitBlock.setIcon(successfulShotIcon);
        successfulHits++;
    }
    System.out.println("Accuracy: " + successfulHits + "/"
            + totalHits);
}
```

εικόνα 45 *- ενημέρωση του δικού μας grid του αντιπάλου*

Μέθοδος mouseClicked() :

Override μέθοδος, που κληρονομήσαμε από το **MouseListener interface** και διαχειρίζεται όλες τις αλληλεπιδράσεις με το ποντίκι. Όλες μας οι κινήσεις πάνω στο grid, όταν προετοιμαζόμαστε αλλά και όταν είμαστε στη διαδικασία των βολών γίνονται με το ποντίκι, και υπάρχουν συνολικά 5 μέθοδοι που διασφαλίζουν την σωστή λειτουργία. Μια μεταβλητή boolean με όνομα "**locked**", διασφαλίζει την περίπτωση όπου αν βρισκόμαστε στη διαδικασία μάχης, οι ίδιες αλληλεπιδράσεις που ίσχυαν όταν είχαμε το στήσιμο των πλοίων, να μην ισχύουν και τώρα. Όσο η μεταβλητή είναι true, ο κώδικας καταλαβαίνει πως είμαστε στη διαδικασία του στησίματος (place).

Ο κώδικας διαχειρίζεται την περίπτωση που το ποντίκι, έχοντας ήδη ο παίκτης επιλέξει διαθέσιμο πλοίο από τη λίστα του, περάσει πάνω από το grid, αναπαριστώντας την αιώρηση (**hover**). Τότε καλείτε η mouseEntered(), και αυτό είναι φυσιολογικό καθώς όπως αφήνει να εννοηθεί και το όνομα της μεθόδου απλά το ποντίκι μπαίνει σε μια περιοχή, δεν γίνεται click ή οτιδήποτε άλλο. Το ποντίκι περνά πάνω από την περιοχή με τα blocks και αυτά αλλάζουν χρώμα με τον προκαθορισμένο, από τις προηγούμενες μεθόδους, τρόπο. Αντίστοιχα, όταν ο κέρσορας του ποντικιού περάσει έξω και απομακρυνθεί από ένα στοιχείο του grid (block), αυτό πάλι επανέρχεται στην προηγούμενη κατάστασή του, και αυτό εκτελείτε από την mouseExited().

Έχουμε μια μεταβλητή boolean με όνομα **horizontal**, όπου κάθε φορά που λαμβάνει την τιμή true, τότε ο κώδικας ορίζει την ακέραια μεταβλητή **orientation** (κατεύθυνση) σε 3, που σημαίνει για εμάς οριζόντια. Το 6, σημαίνει κάθετα, όταν αντίστοιχα η hori-

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

zontal γίνει false. Πήραμε τη λογική από τη φορά των ωροδεικτών, και μας φάνηκε ιδιαίτερα χρήσιμη. Δημιουργήθηκε το πρόβλημα του πως θα αλλάζουμε γρήγορα, κατά τη διάρκεια του στησίματος, το orientation των πλοίων. Το λύσαμε με το δεξί κλικ, και κάθε φορά που το πατάμε το orientation με τη σειρά του αλλάζει από 3 σε 6 και αντίστροφα.

```
} else {
    if (e.getButton() == MouseEvent.BUTTON3) {
        mouseExited(e);
        horizontal ^= true;
        if (horizontal) {
            orientation = 3;
            mouseEntered(e);
        } else {
            orientation = 6;
            mouseEntered(e);
        }
        alliesBoard.validate();
```

εικόνα 46 *- δεξί κλικ και enter/exit*

Αν όμως το κλικ είναι αριστερό, τότε καλούνται όλες εκείνες οι απαραίτητες μέθοδοι όπου μας στήνουν το πλοίο στα ελεύθερα blocks και στην κατεύθυνση που επιλέξαμε.

```
    } else {
        switch (orientation) {
            case (3):
                if (coords[1] < (gridColumns - (shipBlocksNumber - 1))
                        && !warshipBlocksList.contains(currentWarship)) {
                    if (checkCollision()) {
                        battleFormations(false, false);
                        initiateGame();
                    }
                }
                break;
            case (6):
                if (coords[0] < gridRows - (shipBlocksNumber - 1)
                        && !warshipBlocksList.contains(currentWarship)) {
                    if (checkCollision()) {
                        battleFormations(false, false);
                        initiateGame();
                    }
                }
                break;
        }
        alliesBoard.validate();
```

εικόνα 47 *- αριστερό κλικ, τοποθέτηση πλοίων*

Πρέπει με τη σειρά βέβαια πρώτα να κληθεί κάθε μέθοδος που μας διασφαλίζει ότι το πολεμικό πλοίο μπορεί όντως να μείνει εκεί που επιλέξαμε και να συνεχίσουμε στο στήσιμο του επόμενου (δλδ έλεγχος collision, κλπ). Παρατηρούμε πως καλούνται οι μέθοδοι battleFormations() και initiateGame() για κάθε περίπτωση orientation, και γίνεται έλεγχος κολλήματος πρώτα. Μέσα στις if() γίνεται ένας μακροσκελής έλεγχος. Με την βοήθεια των **coords** (συντεταγμένες) και με μεταβλητές πίνακα που κρατάμε τα blocks των πλοίων γίνεται λεπτομερής έλεγχος για το αν το μέγεθος του πλοίου (**total blocks**), χωρά επάνω από την περιοχή πάνω από όπου το αιωρούμε (**hovering**).

Τέλος, η εντολή validate(), όπου καλείτε συνεχώς κατά τον κώδικα, μας βοηθά να "ξαναβάψουμε" θα έλεγε κανείς τα γραφικά αντικείμενά μας στον container. Στην παρούσα περίπτωση στο allyBoard, δηλαδή τον χώρο μας όπου τοποθετούμε τα δικά μας πολεμικά πλοία.

Γυρνάμε πάλι στην αρχή της μεθόδου για να εξηγήσουμε την πρώτη if() που συναντούμε. Αφορά την περίπτωση όπου το παιχνίδι έχει ξεκινήσει και δεν έχει καμία σχέση με το στήσιμο, όπως αφορά όλος ο κώδικας παρακάτω. Αφού λοιπόν η **locked** είναι false μπαίνουμε στην if() μέσα. Έχει να κάνει με τις βολές, δηλαδή την αλληλεπίδραση με το grid, όταν επιχειρούμε click στη μεριά του εχθρικού grid ούτως ώστε να πλήξουμε τα πλοία του αντιπάλου. Στέλνουμε "προς τα έξω" επομένως με τη βοήθεια της TCP σύνδεσης τις συντεταγμένες μιας βολής προς τον εχθρό, γι' αυτό και η εντολή [out.println("hit:" + coords[2]);] Μια μεταβλητή ακεραίων, η totalHits συγκρατεί τις συνολικές βολές που έχουν επιχειρηθεί.

```
@Override
public void mouseClicked(MouseEvent e) {
    GenericBlock clickedBlock = (GenericBlock) e.getSource();
    if (gameStarted && !locked) {
        getBlockPosition(clickedBlock);
        out.println("hit:" + coords[2]);
        totalHits++;
        locked = true;
    }
```

εικόνα 48 *- βολή προς τον εχθρό*

Μέθοδος mousePressed() :

Λόγω του ότι αυτή η μέθοδος, όπως μαρτυρά και το όνομά της, δε μας δίνει καμία χρήση, την έχουμε συνδέσει απευθείας με την μέθοδο mouseClicked(). Φυσικά δεν είναι το ίδιο. Αυτή η μέθοδος αφορά την εξής αλληλεπίδραση με το ποντίκι: Χωρίζουμε ένα click στη μέση, και τι μας μένει? Το πάτημα (**press**) και η απελευθέρωση (**release**), και η εντολή αυτή διαχειρίζεται το πρώτο μισό, δηλαδή τη να συμβεί μόνο με την πίεση. Για να κερδίσουμε χρόνο συνδέσαμε τη μέθοδο απ' ευθείας με τις αλληλεπιδράσεις της mouseClicked().

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

```
@Override
public void mousePressed(MouseEvent e) {
    GenericBlock pressedBlock = (GenericBlock) e.getSource();
    if (pressedBlock.isOnShipsList()) {
        mouseClicked(e);
    }
}
```

εικόνα 49 *- η mousePressed()*

Μέθοδος mouseEntered() :

Εξηγήσαμε τις ιδιότητες της overridden (παράκαμψη) μεθόδου mouseEntered() προ ολίγου, καθώς την καλούμε για λόγους εύρυθμης λειτουργίας και μέσα στην mouseClicked(). Μέσα στην switch() της παρούσας μεθόδου εκτελούνται περίπου οι ίδιες εντολές με αυτές στην αντίστοιχη switch() της mouseClicked().

Όπως και στη φάση που εγκαθιστούμε ένα πολεμικό πλοίο επάνω στο grid (αριστερό click), έτσι και τώρα που χρειαζόμαστε μόνο το **hover effect**, είναι αναγκαίο να γίνουν οι ίδιοι έλεγχοι. Δηλαδή έλεγχος αν είμαστε μέσα στα όρια του grid, έλεγχος κολλήματος σε άλλο πλοίο (**collision detection**), κλπ.

```
            switch (orientation) {
                case (3):
                    if (coords[1] < (gridColumns - (shipBlocksNumber - 1))
                            && !warshipBlocksList.contains(currentWarship)) {
                        if (checkCollision()) {
                            battleFormations(true, false);
                        }
                    }
                    break;
                case (6):
                    if (coords[0] < gridRows - (shipBlocksNumber - 1)
                            && !warshipBlocksList.contains(currentWarship)) {
                        if (checkCollision()) {
                            battleFormations(true, false);
                        }
                    }
                    break;
            }
            alliesBoard.validate();
        }
```

εικόνα 50 *- έλεγχος και εκκίνηση του hover effect*

Οι παρακάτω τρεις γραμμές κώδικα μέσα στην if(), φροντίζουν να αλλάζουν το γραφικό του κέρσορα σε στόχο, κάθε φορά που περνά το ποντίκι μας επάνω από το εχθρικό grid. Δηλαδή όταν είναι να εκτελέσουμε βολή προς τον αντίπαλο, και εισάγουμε το ποντίκι μας πάνω από κάποιο block, στο αντίπαλο grid, αντί για βέλος έχουμε ένα γραφικό στόχου. Είναι μια παράμετρος που μπορεί να αλλάξει και αυτή, ανάλογα τον παίχτη που έχουμε κάθε φορά, με τον ίδιο κώδικα. Θα μπορούσαμε για τον παίκτη **child** να έχουμε π.χ. ένα ξίφος πειρατικό, αλλά σε αυτήν την έκδοση του Battleship Game το έχουμε αφήσει ίδιο για όλους τους παίκτες.

```
if (enteredBlock.getParent().equals(enemyBoard)) {
    Cursor targetCursor = playerValues.getToolkit().createCustomCursor(
            playerValues.getTargetIcon(), cursorHotSpot, "Cursor");
    enemyBoard.setCursor(targetCursor);
}
```

εικόνα 51 *- αλλαγή του κέρσορα σε στόχο*

Μέθοδος mouseExited() :

Φυσικά αφού το ποντίκι περάσει πάνω από ένα block και απομακρυνθεί και δεν επιλεγεί σαν θέση για να τοποθετήσουμε πολεμικό πλοίο, το ίδιο θα πρέπει να επανέλθει στην προηγούμενη κατάστασή του. Αναφέραμε και προηγουμένως για το τι γίνεται όταν περάσει το ποντίκι κάνοντας **hovering**, επάνω από ένα block και δεν γίνει κάποια άλλη αλληλεπίδραση.

Να αναφέρουμε εδώ πως η battleFormations(), που καλείτε συχνά ανάμεσα στις overridden μεθόδους, αν παρατηρήσει κανείς καλείτε συχνά με διαφορετικές τιμές σε κάθε περίπτωση για τα 2 boolean ορίσματα. Λογικό, εφόσον όπως είπαμε είναι μια πολύ ζωτικής σημασίας μέθοδος η οποία εκτελεί πολλαπλές διεργασίες, και μας είναι συνεχώς απαραίτητη. Διαφορετικές είναι οι εργασίες του κάθε φορά ανάλογα από πού καλείτε, και έτσι τα ορίσματα δίνουν στην ίδια να αντιληφθεί ποια μέθοδος την έχει ανάγκη κάθε φορά. Στη συγκεκριμένη περίπτωση, η σειρά false και true στα ορίσματα δίνει στην battleFormations() την πληροφορία ότι την καλεί η mouseExited(). Κάθε μέθοδος έχει τη δική της λοιπόν μοναδική σειρά στα ορίσματα, όταν την καλεί.

4.3. Το API package:

Στο πακέτο **gr.epp.thesis.api** θα βρει κανείς όλες εκείνες τις κλάσεις που γενικοποιούν στο σύνολό τους όλα εκείνα τα αντικείμενα και τις μεταβλητές που χρειάζονται για να δομηθεί τελικά το παιχνίδι. Θα μπορούσαμε να πούμε πως στο σύνολό τους ορίζουν όλους εκείνους του κανόνες που πρέπει να πληρεί ένας τύπος παίκτη που θέλουμε να εισάγουμε στο παιχνίδι. Δηλαδή, οι τύποι: **Adult**, **Child**, **Admiral** που ετοιμάσαμε εμείς για αυτή την έκδοση παιχνιδιού, όντως καλύπτουν τα ερωτήματα που τίθενται. Οι κλάσεις, που δημιουργούνται αμέσως μόλις γίνει επιλογή παίκτη, στο σύνολο τους κάνουν **extend** στιγμιότυπα των κλάσεων του API μας.

Στην παρούσα έκδοση του παιχνιδιού Ναυμαχία, εμείς δημιουργήσαμε 3 κλάσεις για το **custom API** μας. Την **GenericBlock**, την **GenericLabel**, και την **GenericValues**, και κάθε φορά που εισάγεται νέος τύπος παίκτη πρέπει αντίστοιχα να ικανοποιούν τα αιτήματα αυτών των τριών κλάσεων, όπου θα δούμε παρακάτω τι ακριβώς "ζητούν".

Τι σημαίνει αυτό ακριβώς? Σημαίνει πως για κάθε παίχτη, υποχρεωτικά οφείλουμε να χτίσουμε 3 νέες κλάσεις που θα κάνουν extend αυτές τις κλάσεις (GenericBlock,

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

GenericLabel, και GenericValues). Αν παρατηρήσει κανείς τον κώδικα όντως συμβαίνει αυτό, και αν επιχειρήσει να δημιουργήσει έναν νέο παίκτη, αυτόματα οφείλει να τις δημιουργήσει με συγκεκριμένα ονόματα. Πρώτα το όνομα που έδωσε στον τύπο παίκτη και ύστερα κολλητά το είδος της κλάσης που θα κάνει προφανώς extend. Αν για παράδειγμα αποφασίσει, σε μια νέα version του παιχνιδιού, να εισάγει τον τύπο παίκτη **veteran**, θα δημιουργήσει αμέσως τις κλάσεις **VeteranBlock**, **VeteranLabel** και **VeteranValues**. Υπάρχει λόγος που οριοθετούμε έτσι την ονομασία των κλάσεων, αλλιώς δε θα μπορούσαν να κληθούν. Η ιδιότητα του κώδικα να εκτελείτε όπως είναι, χωρίς παραλλαγές, για οποιοδήποτε τύπο παίκτη που εισάγουμε έχει αυτήν την απαίτηση στην ονομασία κλάσεων.

4.3.1. Η κλάση GenericValues:

Το παιχνίδι συναντά μεγάλο αριθμό μεταβλητών, είτε αριθμών, είτε εικόνων και διάφορων άλλων. Προκειμένου να τονίσουμε τις υπαρκτές διαφορές μεταξύ διαφορετικών τύπων παικτών, αλλά και προκειμένου να αποδείξουμε την αξία του **πολυμορφισμού** που έχουμε επιτύχει θα έπρεπε να μπορούμε να έχουμε πολλές αξίες και μεταβλητές που θα ρυθμίζονται ανάλογα στο προφίλ του κάθε τύπου παίκτη. Π.χ. για τον τύπο child έχουν επιλεγεί πιο ζωηρά και παιδικά χρώματα, ενώ οι ίδιες μεταβλητές που αφορούν στην προκειμένη περίπτωση τα χρώματα, για τον παίκτη adult έχουν πιο ρεαλιστικά (πολεμικά) χρώματα.

Ένα πλήθος μεταβλητών λοιπόν είναι απαραίτητο, που με την κατάλληλη ονομασία μας δίνουν να καταλάβουμε περί τίνος πρόκειται. Μερικά από αυτά είναι: gridRows (=γραμμές πλέγματος), frameHeight (=ύψος παραθύρου), waterIcon (=εικόνα για τη θάλασσα), missedShotIcon (=εικόνα αποτυχημένης βολής), και πολλά άλλα.

Στον constructor της κλάσης αυτής οριοθετούμε μερικές μεταβλητές ως προκαθορισμένες (by default) για όλους του παίκτες. Όπως βλέπουμε και στην εικόνα παρακάτω, λόγου χάρη το χρώμα της θάλασσας είναι ίδιο για όλους τους παίκτες εφόσον έχει οριστεί στον constructor της κλάσης αυτής. Ή οι διαστάσεις του παραθύρου, frameWidth και frameHeight, παρατηρούμε πως είναι δηλωμένη (καρφωτά) για όλους τους παίκτες. Όμως ακόμα και αυτές οι μεταβλητές μπορούν να αλλάξουν, δηλαδή υπάρχει ο απαραίτητος μηχανισμός ας πούμε για να βάλεις άλλη εικόνα θάλασσας για τον παίκτη adult. Γίνεται δια μέσου **Getters & Setters**, όπου θα αναφερθούμε και παρακάτω.

```
public GenericValues(int gridRows, int gridColumns) {
    this.frameWidth = 525;
    this.frameHeight = 1050;
    this.toolkit = Toolkit.getDefaultToolkit();
    this.gridRows = gridRows;
    this.gridColumns = gridColumns;
    this.seaColor = Color.CYAN.darker();
    this.allyShipListBackgroundColor = Color.LIGHT_GRAY.darker();
    this.enemyShipListBackgroundColor = Color.DARK_GRAY.darker();
    this.allyShipListBorder = new LineBorder(Color.GREEN.darker(), 1, false);
    this.enemyShipListBorder = new LineBorder(Color.RED.darker(), 1, false);
    this.waterIcon = new ImageIcon("graphics/water.gif");
    this.targetIcon = this.toolkit.getImage("graphics/sword.gif");
    this.successfulShotIcon = new ImageIcon("graphics/fire.gif");
    this.missedShotIcon = new ImageIcon("graphics/miss.gif");
}
```

εικόνα 52 *- constructor της GenericValues*

Η μέθοδος getGridPieces(), μας βοηθά να τοποθετούμε με τη σωστή σειρά τα κομμάτια – εικόνες που χρειαζόμαστε επάνω στο grid. Έχει 2 περιπτώσεις για κάθε πλοίο, κανονικό και βυθισμένο (sinked). Τροφοδοτούμε την μέθοδο με 4 ορίσματα, 3 ακέραιους (int) και μια boolean. Η πρώτη αφορά το συνολικό αριθμό των blocks που απαιτεί ένα πλοίο (που αυτόματα δηλώνει στον κώδικα για ποιο πλοίο μιλάμε), το τρέχων block (κάθε μέρος του πλοίου, δηλαδή κάθε block, έχει δικό του μοναδικό γραφικό όπου όλα μαζί σχηματίζουν το πλοίο), την διεύθυνση (orientation), καθώς και ένα true/false αν είναι βυθισμένο ή όχι. Ανάλογα, όπως παρατηρούμε και στον κώδικα, η boolean μεταβλητή που εξετάζεται στην if() ορίζει και ποια εντολή θα εκτελεστεί, όπου θα τοποθετήσει σωστά τα πλοία στο πλέγμα μας.

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

```
/**
 * Method needed for the placement of warships on the grid. Every
 * orientation.
 *
 * @param shipBlocksNumber Number of blocks of the current ship
 * @param currentBlock
 * @param orientation 3 - horizontal, 6 - Vertical
 * @param sinked
 * @return
 */
public ImageIcon getGridPieces(int shipBlocksNumber, int currentBlock,
        int orientation, boolean sinked) {
    if (sinked) {
        System.out.println("SINK!");
        return (new ImageIcon("graphics/gridPieces/Sink_" + shipBlocksNumber
                + "_" + currentBlock + "_" + orientation + ".gif"));
    } else {
        return (new ImageIcon("graphics/gridPieces/" + shipBlocksNumber
                + "_" + currentBlock + "_" + orientation + ".gif"));
    }
}
```

εικόνα 53 *- μέθοδος getGridPieces()*

Ύστερα ακολουθούν πολλές γραμμές κώδικα όπου πρόκειται για τους **Getters & Setters**. Οι μέθοδοι αυτοί μας βοηθούν να περνάμε νέες τιμές σε μια μεταβλητή, ή να λαμβάνουμε τις υπάρχοντες. Μπορεί να χρειαζόμαστε την τιμή μιας μεταβλητής για να συνεχίσουμε την δουλειά μας, άρα θα την καλέσουμε με τον **getter** της, και αντίστοιχα αν θέλουμε να την αλλάξουμε θα χρησιμοποιήσουμε τον **setter** της, και μαζί τη νέα τιμή, όπου η τελευταία περνά σαν όρισμα με το που εκτελέσουμε την εντολή του set, και θα αντικαταστήσει την υπάρχουσα. Όπως προαναφέραμε, μπορούμε να αλλάξουμε και τα προκαθορισμένα για όλους του παίκτες ορίσματα, που είναι στον constructor της κλάσης αυτής. Θα χρησιμοποιήσουμε τέτοιες μεθόδους για αυτή τη δουλειά, και κάθε μεταβλητή έχει ένα Getter και έναν Setter.

4.3.2. Η κλάση GenericBlock:

Όπως αφήνει να εννοηθεί και το όνομά της, η κλάση αυτή είναι μια γενικευμένη κλάση που περιγράφει τις ιδιότητες ενός block, την οποία θα κάνουν αργότερα extend οι αντίστοιχες κλάσεις block κάθε τύπου παίκτη. Η κλάση αυτή, με τη σειρά της, κάνει extend το JButton, και αυτό μας δίνει μια πολύ σημαντική πληροφορία και αποκαλυπτική! Όλα τα blocks που θα συναντήσουμε στο grid τελικά δεν είναι κάτι άλλο από κουμπιά (**JButtons**), άρα και όλα τα interactions τους (**αλληλεπιδράσεις**) είναι χαρακτηριστικά ενός κουμπιού του swing.

Στον πρώτο της constructor, όπου και μας βοηθά στο να στήσουμε το πλέγμα του

παιχνιδιού καθώς και με ότι αυτό συνεπάγεται (χρώματα, εικόνες, μεγέθη) βλέπουμε την υποχρεωτική δήλωση 2 μεθόδων. Δηλαδή για τις κλάσεις που θα κάνουν extend την παρούσα, θα είναι υποχρεωτικό να ικανοποιηθούν οι απαιτήσεις των 2 αυτών μεθόδων, της initializeValues() και η initializeGridBlocks(). Η πρώτη υποχρεώνει την δήλωση και παραμετροποίηση των μεταβλητών κάθε παίκτη, δηλαδή την αρχικοποίηση και χαρακτηρισμό κάποιον μεταβλητών, ώστε το παιχνίδι να μπορέσει να χτιστεί. Η επόμενη κτίζει το πλέγμα σύμφωνα με τις τιμές και τις παραμέτρους κάθε τύπο παίκτη. Προετοιμάζει τα χρώματα, που διαφέρουν ανά περίπτωση, τις κατάλληλες εικόνες, καθώς και φροντίζει να υπάρχει ο κατάλληλος τρόπος διαχωρισμός κάθε τύπου block, μιας και έχουμε φίλια blocks (εκεί που τοποθετούμε τα πολεμικά πλοία μας), αλλά και εχθρικά (εκεί που εκτελούμε βολές.

```
/**
 * Constructor needed for the seaBlocks Components.
 */
public GenericBlock() {
    initializeValues();
    initializeGridBlocks();
}
```

εικόνα 54 *- ο πρώτος constructor της GenericBlock*

Υπάρχει και δήλωση δεύτερου constructor όπου επιδέχεται και 2 ορίσματα (int και boolean). Αυτός μας βοηθά να χτίσουμε τη λίστα μας, στα δεξιά, με τα διαθέσιμα πολεμικά μας πλοία, πριν τα στήσουμε στο πλέγμα (grid). Πάλι χρειαζόμαστε και εδώ να κληθούν υποχρεωτικά 2 μέθοδοι, η initializeValues(); και η initializeShipsList();. Η πρώτη πάλι υποχρεώνει την δήλωση και παραμετροποίηση των μεταβλητών, δηλαδή την αρχικοποίηση των μεταβλητών. Η δεύτερη προετοιμάζει και χτίζει τη λίστα που προαναφέραμε. Μια λίστα έχουμε διαθέσιμη εμείς, με τα δικά μας πλοία και μια ο αντίπαλος με τα πλοία που αντιμετωπίζουμε.

```
/**
 * Constructor needed for the list of available warships.
 */
public GenericBlock(int index, boolean player) {
    initializeValues();
    this.index = index;
    this.player = player;
    initializeShipsList();
}
```

εικόνα 55 *- ο δεύτερος constructor της GenericBlock*

Ακολουθούν οι abstract μέθοδοι (γενικευμένες – αφαιρετικές μέθοδοι) που αναφέραμε στους constructors. Είναι υποχρεωτικό το χτίσιμο τους, καθώς είναι υποχρεωτικό να ληφθούν υπόψη από της κλάσεις που κάνουν extend την GenericBlock.

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

```
/**
 * Initializing current player's values.
 */
public abstract void initializeValues();

/**
 * Initializing the Grid. Preparing seaBlocks. Background Color & se
 * Icon.
 */
public abstract void initializeGridBlocks();

/**
 * Initializing the ships list. Preparing available warships.
 */
public abstract void initializeShipsList();
```

εικόνα 56 *- οι abstract μέθοδοι της GenericBlock*

Επίσης και εδώ, προκειμένου να έχουμε άμεση πρόσβαση στις μεταβλητές μας και στην παραμετροποίηση αυτών, ακολουθούν Getters και Setters.

4.3.3. Η κλάση GenericLabel:

Μια ακόμα γενικευμένης φύσης κλάσης όπου έχει μόνο αισθητική σημασία σε όλο το project μας. Παρ' όλα αυτά με τη σειρά της και αυτή φροντίζει να υποχρεώσει τις κλάσεις που την επεκτείνουν (**extend**) να ορίσουν και να αρχικοποιήσουν τις μεταβλητές και τους όρους που χρειάζονται. Κυρίως διαχειρίζεται όσα panel διαθέτουν εικόνες και γραφικά, που όπως είναι λογικό λόγω του πολυμορφισμού που έχουμε εισάγει στο παιχνίδι κάθε παίκτης έχει διαφορετικές εικόνες, και όχι μόνο (διαφορετικά χρώματα, αποχρώσεις κλπ.). Για τον παίκτη child π.χ. έχουμε ανάγκη να χρησιμοποιήσουμε άλλου τύπου εξώφυλλο (cover) στον κορμό του παιχνιδιού απ' ότι στον παίκτη admiral. Για μας, κάθε νέος παίκτης που θα εισάγεται στο παιχνίδι θα πρέπει να έχει ορίσει τις εικόνες του που θα εμφανίζονται στα labels, χρώματα borders κλπ.

Ο πρώτος constructor υποχρεώνει την εκτέλεση της abstract μεθόδου initializeValues(); όπου οι μεταβλητές που καθορίζει ύστερα αυτή ορίζουν το κεντρικό panel, το εξώφυλλο δηλαδή, ανά περίπτωση παίκτη, και ο δεύτερος constructor υποχρεώνει την εκτέλεση πάλι της initializeValues(); αλλά και της initializePlayerLabel(); όπου και αυτές με τη σειρά του αφού ενημερωθούν για τις ισχύοντες μεταβλητές ανά παίκτη αρχικοποιούν στη συνέχεια γραφικά και εικόνες που μας χρειάζονται για να τοποθετήσουμε στα panel των λιστών με τα πολεμικά μας πλοία, αλλά και του αντιπάλου. Ακολουθεί το σώμα των abstract μεθόδων.

```
/**
 * Constructor for the decor panel of a specific player.
 */
public GenericLabel() {
    initializeValues();
}

/**
 * Constructor for the player labels above ships list of a specific
 */
public GenericLabel(boolean player) {
    initializeValues();
    this.player = player;
    initializePlayerLabel();
}

/**
 * Initializing current player's values.
 */
public abstract void initializeValues();

/**
 * Initializing current player's label.
 */
public abstract void initializePlayerLabel();
```

εικόνα 57 *- το σώμα της κλάσης GenericLabel*

4.4. Η κλάση BattleshipMain:

Από την κλάση αυτή ξεκινά την εμπειρία του ο κάθε παίκτης που συνδέεται σε μια συνεδρία του παιχνιδιού. Κάθε φορά που εισάγεται δηλαδή ένας παίκτης, τρέχει και ένα instance της κλάσης αυτής.

Στην αρχή υπάρχει ο απαραίτητος κώδικας που μας παρουσιάζει το πρώτο panel όπου και καλούμαστε να επιλέξουμε τύπο παίκτη. Μέσα στο constructor αυτόν υπάρχουν οι κατάλληλες δηλώσεις και μεταβλητές που μας φτιάχνουν το αρχικό παράθυρο λοιπόν, το πρώτο παράθυρο που βλέπεις κανείς όταν τρέχει το παιχνίδι. Μια λίστα με επιλογές υπάρχει, όπου επιλέγει κανείς ανάμεσα σε 3 είδη παικτών, τουλάχιστον στην έκδοση του παιχνιδιού αυτού. Μπορούν πολύ εύκολα να επινοηθούν και άλλα είδη παίκτη (τυφλός, με ηχητικές υποβοηθήσεις παραδείγματος χάριν). Οι απαραίτητοι Listeners έχουν και σε αυτή την κλάση εγκατασταθεί επάνω στα components ούτως ώστε να πυροδοτήσουν στη συνέχεια τα απαραίτητα συμβάντα. Παρουσιάσουμε νωρίτερα πως είναι το περιβάλλον εκκίνησης του παιχνιδιού καθώς και η επιλογή παίκτη με παράδειγμα εικόνας.

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

```
public BattleshipMain() {
    startingFrame.setVisible(true);
    startingFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    startingFrame.setLocationRelativeTo(null);
    startingFrame.setSize(500, 334);
    startingFrame.setResizable(false);
    startingFrame.add(backround);

    backround.setLayout(new GridLayout(1, 2, 20, 20));
    backround.add(tempPanelWest);
    backround.add(tempPanelEast);

    tempPanelWest.setOpaque(false);
    setTempCells(tempPanelWest, 3, false);
    tempPanelWest.add(playerTitle);
    tempPanelWest.add(playerTypeBox);
    tempPanelEast.setOpaque(false);
    setTempCells(tempPanelEast, 3, false);
    tempPanelEast.add(observerTitle);
    setTempCells(tempPanelEast, 1, true);
    tempPanelEast.add(observerButton);

    playerTitle.setFont(new Font("Verdana", Font.BOLD, 14));
    playerTitle.setForeground(Color.BLACK);

    observerTitle.setFont(new Font("Verdana", Font.BOLD, 14));
    observerTitle.setForeground(Color.BLACK);
```

εικόνα 58 *- ο πρώτος constructor της BattleshipMain()*

Στην περίπτωση πού είχαμε observers στο παιχνίδι (αναφέρουμε στο κεφάλαιο "Επεκτάσεις της πτυχιακής") ο παρακάτω κώδικας στην ουσία δημιουργεί το περιβάλλον του παρατηρητή, όπου μπαίνει μόνο για να είναι θεατής σε μια ήδη εξελισσόμενη Ναυμαχία, σύμφωνα με τα πρότυπα (γραφικό περιβάλλον, κλπ) του παίκτη adult. Βέβαια δεν έχει καμία απολύτως σχέση με interactions. Δηλαδή κανένας listener δεν "ακούει" τις επιλογές του.

```
        @Override
        public void actionPerformed(ActionEvent e) {
            setCurrentPlayerType("observer");
            currentPlayer = "Adult";
            startingFrame.dispose();
            try {
                tempClass = Class.forName("gr.epp.thesis." + currentPlayer +
                playerValues = (GenericValues) tempClass.newInstance();
            } catch (ClassNotFoundException ex) {
                Logger.getLogger(ChildBlock.class.getName()).log(Level.SEVER
            } catch (InstantiationException ex) {
                Logger.getLogger(BattleshipMain.class.getName()).log(Level.S
            } catch (IllegalAccessException ex) {
                Logger.getLogger(BattleshipMain.class.getName()).log(Level.S
            }
            Thread thread = new Thread(new BattleshipMain(currentPlayerType,
            thread.start();
        }
    });
}
```

εικόνα 59 *- διαμόρφωση περιβάλλοντος παρατηρητή*

Με τον δεύτερο constructor, με τη σειρά τους τώρα, καλούνται όλες εκείνες οι απαραίτητες μεταβλητές όπου μας αρχικοποιούν και μας εκκινούν τα κυρίως frame ανά παίκτη όπου μέσα τους, όπως διαβάζουμε και από τον κώδικα, βρίσκονται πολλά panels, buttons, κλπ, με τα οποία ο παίκτης αλληλεπιδρά και ουσιαστικά παίζει το παιχνίδι της Ναυμαχίας. Υπάρχει ένα κύριο frame όπου συγκεντρώνει με τη σειρά του άλλα panels, μέσα στα οποία ίσως βρίσκονται άλλα panels ή labels ή buttons και διάφορα άλλα components. Όλα τοποθετούνται και οργανώνονται όπως ορίζει ο εκάστοτε layouter σε κάθε container κάθε φορά. Ο παρακάτω κώδικας είναι υπεύθυνος για το κύριο γραφικό μέρος του κορμού του παιχνιδιού πάνω στο οποίο εξελίσσεται στα μάτια μας όλο το παιχνίδι, μέχρι να χάσουμε ή νικήσουμε.

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

```
/**
 * Main Game:
 */
public BattleshipMain(String currentPlayer, GenericValues playerValues)
    this.playerValues = playerValues;
    masterFrame.setSize(this.playerValues.getFrameWidth(), this.playerV
    masterFrame.setResizable(false);
    masterFrame.setVisible(true);
    masterFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    masterFrame.setLocationRelativeTo(null);
    masterFrame.setLayout(new GridLayout(3, 1, 0, 5));
    masterFrame.setBackground(Color.WHITE);
    masterFrame.add(upPanel);
    upPanel.setBackground(Color.WHITE);
    masterFrame.add(decorPanel);
    decorPanel.setBackground(Color.WHITE);
    masterFrame.add(downPanel);
    downPanel.setBackground(Color.WHITE);
    upPanel.setLayout(new BorderLayout(10, 0));
    upPanel.add(enemyBoard, BorderLayout.CENTER);
    enemyBoard.setLayout(new GridLayout(this.playerValues.getGridRows()
    downPanel.setLayout(new BorderLayout(10, 0));
    downPanel.add(allyBoard, BorderLayout.CENTER);
    allyBoard.setLayout(new GridLayout(this.playerValues.getGridRows(),
    allyShipsListPanel = new JPanel(new GridLayout(playerValues.getShip
    enemyShipsListPanel = new JPanel(new GridLayout(playerValues.getShi
    downPanel.add(allyShipsListPanel, BorderLayout.EAST);
    upPanel.add(enemyShipsListPanel, BorderLayout.WEST);
    masterFrame.validate();
```

εικόνα 60 *- ο δεύτερος constructor της BattleshipMain()*

Η σύνδεση tcp γίνεται, ο παίκτης στέλνει και λαμβάνει πλέον strings με τον server, που ήδη τρέχει, και κατ' επέκταση με το άλλον παίκτη όπου και αυτός με τη σειρά του έχει συνδεθεί στον server. Χρησιμοποιείτε όπως βλέπουμε το port 1501. Και στη συνέχεια αρχικοποιείτε η κλάση GameControl που διαχειρίζεται στην ουσία όλο το παιχνίδι όπως αναφέραμε προηγουμένως.

```
        int portNumber = 1501;
        String host = "localhost";
        System.out.print("Connect with " + host + " in port " + portNumber +
        try {
            clientSocket = new Socket(host, portNumber);
            System.out.println("Connected");
//              gameControl.setSocket(clientSocket);
        } catch (UnknownHostException ex) {
            Logger.getLogger(BattleshipMain.class.getName()).log(Level.SEVER
        } catch (IOException ex) {
            Logger.getLogger(BattleshipMain.class.getName()).log(Level.SEVER
        }

        gameControl = new GameControl(playerValues, clientSocket, currentPla
```

εικόνα 61 *- σύνδεση νέου client στο server*

Ο κώδικας παρακάτω εφαρμόζει την τεχνική του πολυμορφισμού στο project μας. Δηλαδή όλες εκείνες τις απαραίτητες δηλώσεις και μεταβλητές όπου φροντίζουν να μεταφράζουν τις ανάγκες του κάθε τύπου παίκτη που αποφασίζει να συνδεθεί στο παιχνίδι, διατηρώντας όμως το ίδιο σώμα στον κώδικα. Δηλαδή ο κώδικα δεν αλλάζει σε καμία περίπτωση για την ανά παίκτη περίπτωση, αλλά πάντα μένει σταθερός, και όπως είναι δομημένος εκτελείτε με τη σειρά το ίδιο για όλους τους τύπους παίκτη. Συν τοις άλλοις, όσους τύπους παίκτη και να εισάγουμε, αν καλύψουμε τις υποχρεωτικές ανάγκες σε δηλώσεις και ρυθμίσεις που ορίζει το custom API μας, πάλι ο συγκεκριμένος κώδικα θα μπορεί να μας καλύψει. Με λίγα λόγια δεν πειράζουμε ούτε διαμορφώνουμε καθόλου αυτήν την κλάση.

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

```
try {
    tempClass = Class.
            forName("gr.epp.thesis." + currentPlayer + "Block");
    for (int i = 0; i < this.playerValues.getGridRows()
            * this.playerValues.getGridColumns(); i++) {
        GenericBlock enemySeaBlock = (GenericBlock) tempClass.
                newInstance();
        if (getCurrentPlayerType().equals("gamer")) {
            enemySeaBlock.addMouseListener(gameControl);
            enemyBoard.add(enemySeaBlock);
        }
        GenericBlock mySeaBlock = (GenericBlock) tempClass.newInstance();
        mySeaBlock.addMouseListener(gameControl);
        allyBoard.add(mySeaBlock);
    }

    tempClass = Class.
            forName("gr.epp.thesis." + currentPlayer + "Label");
    GenericLabel decorLabel = (GenericLabel) tempClass.newInstance();
    decorPanel.add(decorLabel);
    Constructor playerLabelsConstructor = tempClass.getConstructor(
            boolean.class);
    GenericLabel enemyPlayerLabel
            = (GenericLabel) playerLabelsConstructor.newInstance(false);
    GenericLabel playerLabel = (GenericLabel) playerLabelsConstructor.
            newInstance(true);

    tempClass = Class.
            forName("gr.epp.thesis." + currentPlayer + "Block");
    Constructor tempShipConstructor = tempClass.
            getConstructor(int.class, boolean.class);
    enemyShipsListPanel.add(enemyPlayerLabel);
    allyShipsListPanel.add(playerLabel);
    for (int i = 0; i < playerValues.getShipListSize() - 1; i++) {
        GenericBlock enemyWarship = (GenericBlock) tempShipConstructor.
                newInstance(i, false);
        enemyWarship.addMouseListener(gameControl);
        enemyShipsListPanel.add(enemyWarship);
        GenericBlock myWarship = (GenericBlock) tempShipConstructor.
                newInstance(i, true);
        myWarship.addMouseListener(gameControl);
        allyShipsListPanel.add(myWarship);
    }

    gameControl.
            setLateValues(enemyBoard, allyBoard, enemyShipsListPanel);
    (new Thread(gameControl)).start();
```

```
    } catch (NoSuchMethodException | IllegalArgumentException |
            InvocationTargetException ex) {
        Logger.getLogger(BattleshipMain.class.getName()).log(Level.SEVERE,
                null, ex);
    } catch (ClassNotFoundException ex) {
        Logger.getLogger(ChildBlock.class.getName()).log(Level.SEVERE, null,
                ex);
    } catch (InstantiationException ex) {
        Logger.getLogger(ChildBlock.class.getName()).log(Level.SEVERE, null,
                ex);
    } catch (IllegalAccessException ex) {
        Logger.getLogger(ChildBlock.class.getName()).log(Level.SEVERE, null,
                ex);
    }

    masterFrame.validate();
```

εικόνα 62 *- ο κώδικας που εφαρμόζει τον πολυμορφισμό*

4.5. Η κλάση Server:

Σε αυτό το σημείο παρουσιάζουμε την κλάση του server, όπου μπορεί εύκολα να παρατηρήσει κανείς πως διαθέτει απλές λειτουργίες. Ο εξυπηρετητής (**server**) μας ακούει σε ένα port που του έχουμε εμείς ορίσει και αναμένει την προσέγγιση των clients, μέγιστος αριθμός όπου εμείς έχουμε αποφασίσει σε 2, αλλά μπορεί να αυξηθεί. Ή πάλι μπορεί με κάποια παράμετρο που θα έχουμε υπολογίσει νωρίτερα όσο αφορά την ταυτότητα των clients, να καταλαβαίνει πότε κάποιος παίκτης είναι κανονικός διαγωνιζόμενος ή παρατηρητής (**observer**), και αντίστοιχα να ρυθμίζει ποιά strings περνάνε από το ρεύμα στο οποίο έχει ρόλο μεσίτη, και ποια όχι.

Ύστερα όπως παρουσιάζεται και στον κώδικα δημιουργούνται τα sockets για κάθε παίκτη που συνδέεται στο port 1501, και ξεκινούν 2 **threads** (παράλληλες διεργασίες). Ένα για να αναγνώσει τα εισερχόμενα μηνύματα και άλλο ένα για να υπολογίσει τον αριθμό των ήδη συνδεδεμένων κάθε φορά.

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

```
public class Server {

    private static ServerSocket serverSocket = null;
    private static Socket clientSocket = null;
    private static final int maxClientsCount = 2; //Maximum number of conne
    private static final NumbersThread[] threads = new NumbersThread[maxCl
    private static final boolean[] playerTypes = new boolean[threads.lengt

    public static void main(String[] args) {
        int portNumber = 1501;
        int portNumber2 = 1502;

        if (args.length < 1) {
            System.out.println("Now using port number=" + portNumber);
        } else {
            portNumber = Integer.valueOf(args[0]).intValue();
        }
        try {
            //Opens a server socket in ports 1501 (portNumber).
            serverSocket = new ServerSocket(portNumber);
        } catch (IOException ex) {
            System.out.println(ex.getMessage());
        }
```

εικόνα 63 *- τα port της κλάσης Server*

```
    /**
     * Creates a client socket for each connection and starts two threads.
     * One for reading incoming data(NumbersThread) and another for
     * calculating the number of connected clients(ClientThread).
     */
    while (true) {
        try {
            clientSocket = serverSocket.accept();
            System.out.println("User Connected in port: " + portNumber);
            for (int i = 0; i < maxClientsCount; i++) {
                if (threads[i] == null) {
                    (threads[i] = new NumbersThread(clientSocket, threads,
                    break;
                }
            }
        } catch (IOException ex) {
            System.out.println(ex.getMessage());
        }
    }
```

εικόνα 64 *- ξεκίνημα των threads*

5. Μελλοντική Εργασία και Επεκτάσεις

5.1. Model - View - Controller (MVC)

5.1.1. Σκοπός:

Να διαιρέσει ένα συστατικό σε τρία λογικά μέρη: μοντέλο (model), αναπαράσταση (view) και διαχείριση (controller) καθιστώντας ευκολότερη την διαδικασία τροποποίησης κάθε μέρους. ***[13]*** Θα μπορούσε να εισαχθεί αυτή η τεχνική ανάπτυξης στο ήδη υπάρχων παιχνίδι. Είναι ιδιαίτερη πρόκληση με πολλά προσφερόμενα πλεονεκτήματα. Μερικές πληροφορείς προσφέρονται παρακάτω.

5.1.2. Προσέγγιση με ένα παράδειγμα:

Έστω ότι επιθυμούμε να παραστήσουμε ένα ραντεβού με κάποιον γνωστό μας στο προσωπικό ηλεκτρονικό organizer, που κατασκευάζουμε. Πιθανότατα, θα δημιουργούσαμε μια κλάση όπου θα ορίζαμε γενικές πληροφορίες, όπως το όνομα του, το επώνυμο, η επιχείρηση στην οποία εργάζεται κ.ο.κ. Περαιτέρω θα επιθυμούσαμε να παραστήσουμε το ραντεβού αυτό κι οπτικά (visually). Έτσι, θα έπρεπε να προσθέσουμε κώδικα και για αυτό στην κλάση. Τέλος, θα έπρεπε να προσθέσουμε και μια σειρά μεθόδων έτσι ώστε κάθε αλλαγή στο GUI να ανανεώνει κατάλληλα τις πληροφορίες του ραντεβού.

Βέβαια, αυτή η λύση παρουσιάζει ορισμένα προβλήματα:

- Αν και κάποιος μπορεί να ισχυριστεί ότι όλα αυτά τα μέρη αντιπροσωπεύουν το αντικείμενο Contact (ραντεβού) και πρέπει να είναι μαζί, η χρησιμοποίηση μονάχα μιας κλάσης καθιστά τον κώδικα πολύπλοκο και δύσκολα διαχειρίσιμο.
- Η κλάση αυτή δεν είναι εύκολα επεκτάσιμη. Τι θα συνέβαινε αν για παράδειγμα επιθυμούσαμε μια διαφορετική οπτική αναπαράσταση του ραντεβού; Με μονάχα μια κλάση, κάτι τέτοιο δεν μπορεί να πραγματοποιηθεί αποτελεσματικά.

Μια καλύτερη προσέγγιση θα ήταν να διαχωρίσουμε τα τρία αυτά λογικά μέρη σε διαφορετικές κλάσεις. Μια κλάση θα παριστά τις πληροφορίες του ραντεβού, μια την οπτική αναπαράσταση του και μια θα διαχειρίζεται την επικοινωνία μεταξύ του GUI και των πληροφοριών του ραντεβού. Το Pattern αυτό ονομάζεται Model – View – Controller (ή απλούστερα MVC) και είναι ιδιαίτερα χρήσιμο όταν επιθυμούμε να δημιουργήσουμε εύκαμπτα και εύκολα διαχειρίσιμα συστατικά.

5.1.3. Εφαρμογές:

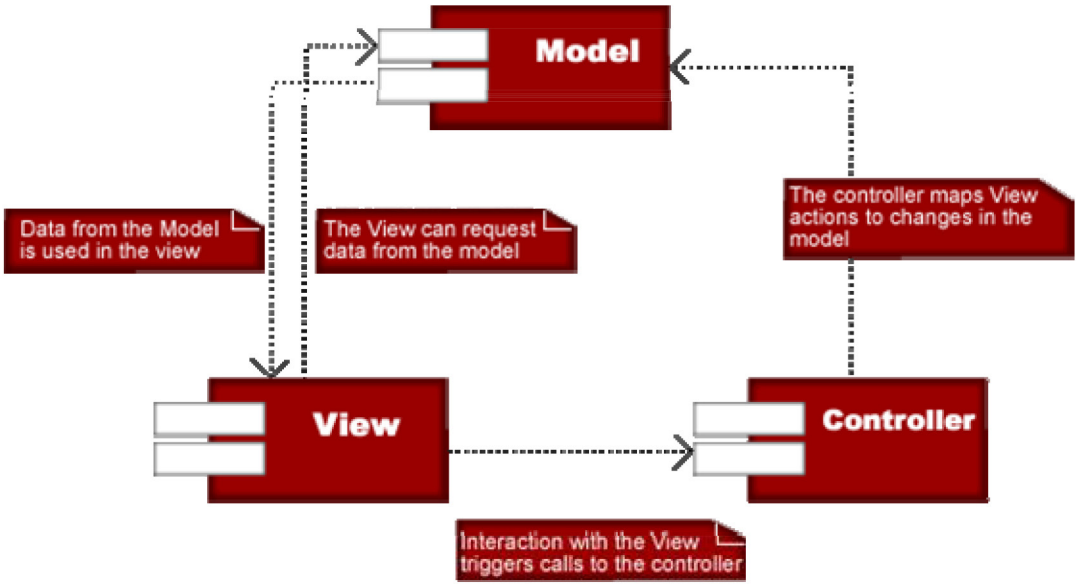
Το Pattern MVC χρησιμοποιείται όταν ένα συστατικό έχει κάποιο από τα ακόλουθα χαρακτηριστικά:

- Είναι πιθανό να δούμε το συστατικό με διαφορετικούς τρόπους. Η εσωτερική αναπαράσταση του αντικειμένου μπορεί να είναι εντελώς διαφορετική από την εξωτερική του (στην οθόνη).
- Η συμπεριφορά ή η αναπαράσταση του συστατικού μπορούν να αλλάξουν

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

δυναμικά.

- Υπάρχουν διαφορετικοί τύποι συμπεριφορών, με την έννοια ότι πολλών ειδών συμπεριφορές μπορούν να υλοποιηθούν για το ίδιο αντικείμενο.
- Πολλές φορές επιθυμούμε να επαναχρησιμοποιήσουμέ ένα τέτοιο συστατικό κάτω από άλλες συνθήκες, με τις λιγότερες δυνατές αλλαγές.



εικόνα 65 *- διάγραμμα MVC*
source: *http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller*

5.1.4. Επεξήγηση:

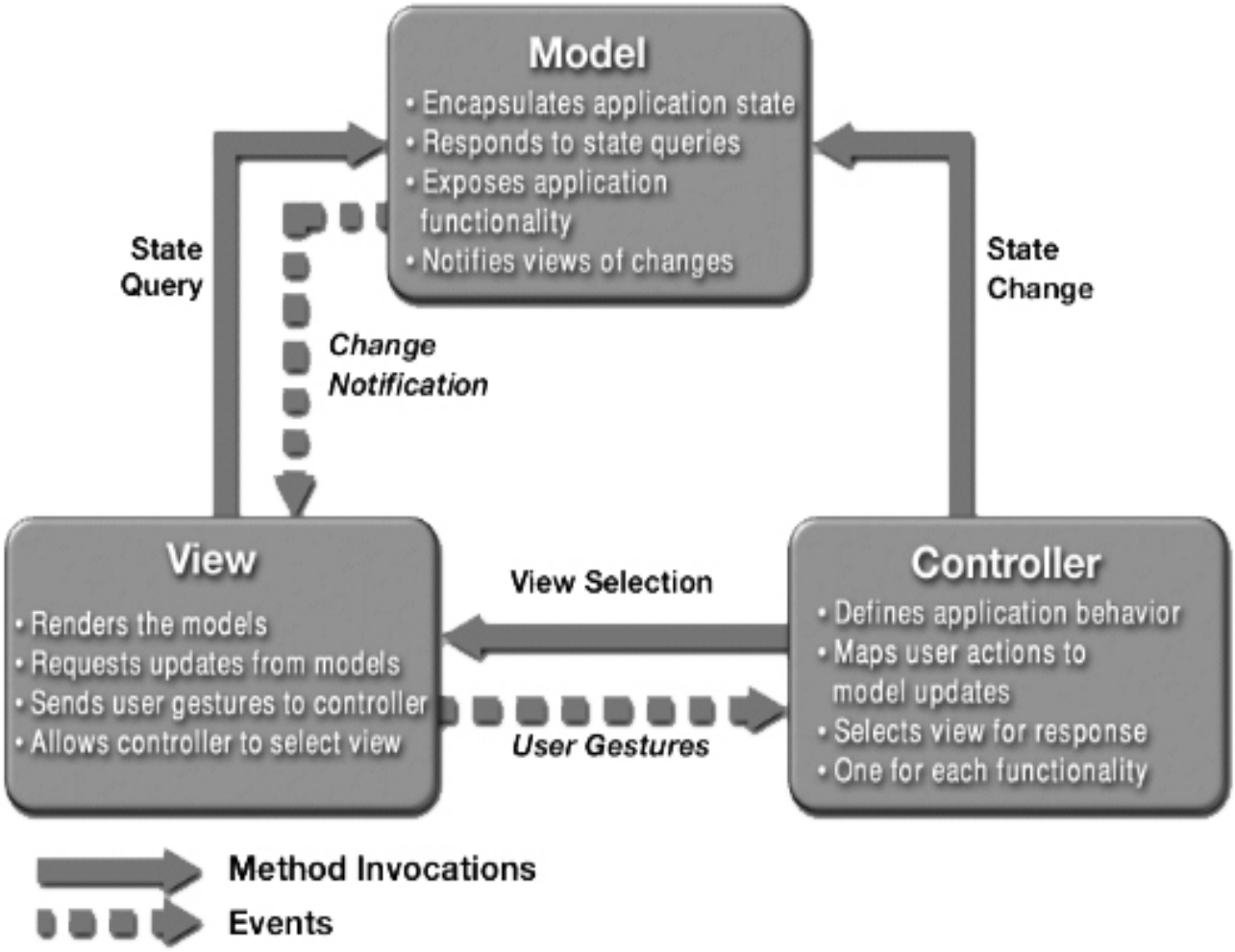
Model:
Το συστατικό αυτό περιέχει μία ή περισσότερες κλάσεις ή Interface, που είναι υπεύθυνα για την υποστήριξη των πληροφοριών του μοντέλου. Η κατάσταση του μοντέλου διατηρείται στα πεδία και τροποποιείται από μεθόδους. Επίσης, συγκρατείται μία αναφορά σε κατάλληλα View, ώστε να ενημερώνονται κάθε φορά που παρατηρούνται αλλαγές.

View:
Κλάσεις και Interface που παρέχουν οπτικές αναπαραστάσεις του συστατικού. Κάθε φορά που παρατηρείται κάποια αλλαγή στην πληροφορίες του Model, το View είναι υπεύθυνο για το αν και πώς θα οπτικοποιήσει αυτήν την αλλαγή.

Controller:
Αυτό το συστατικό διαχειρίζεται τις αλλαγές στο Model. Διατηρεί μια αναφορά στο συστατικό Model που είναι υπεύθυνο για τις αλλαγές, ενώ καλεί μία ή περισσότερες update μεθόδους. Η εντολή αλλαγής μπορεί βέβαια να έρθει και από κάποιο View.

5.1.5. Πλεονεκτήματα και Μειονεκτήματα:

Το Pattern MVC παρέχει συστατικά εύκαμπτα και εύκολα προσαρμόσιμα σε νέες συνθήκες. Αυτή η ευκαμψία μπορεί να χρησιμοποιηθεί είτε στατικά, είτε δυναμικά. Νέες View ή Controller κλάσεις μπορούν να προστεθούν στην εφαρμογή (στατικά) και view ή controller αντικείμενα μπορούν να αλλάξουν καθώς η εφαρμογή εκτελείται (δυναμικά).



εικόνα 66 *- διάγραμμα MVC*
source: *http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller*

5.1.6. Παραλλαγές του Μοντέλου:

Οι παραλλαγές του μοντέλου αφορούν κυρίως διαφορετικές υλοποιήσεις του View:

- *Multiple view targets:*
  Ένα Model μπορεί να παρέχει πληροφορίες σε περισσότερα από ένα View. Αυτό συνήθως είναι χρήσιμο σε GUI υλοποιήσεις.

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389

- *"Look but don't touch":*
  Τα View δεν χρειάζονται πάντοτε κάποιον Controller. Μπορούν να χρησιμοποιηθούν μονάχα για οπτικές αναπαραστάσεις των πληροφοριών του μοντέλου χωρίς να υποστηρίζουν αλλαγές στις πληροφορίες αυτές.

- *Model push vs view pull:*
  Το MVC μπορεί να υλοποιηθεί κατά δύο τρόπους. Το Model μπορεί να στέλνει updates στο view (ή στα views) ή ένα view μπορεί να λαμβάνει πληροφορίες όποτε χρειάζεται από το model. Αυτή η επιλογή την υλοποίηση της σχέσης στο σύστημα.

5.1.7. Σχετικά Μοντέλα:

Τα σχετικά μοντέλα τουMVC είναι τα ακόλουθα:

- *Observer:*
  Το MVC χρησιμοποιεί συχνά το μοντέλο αυτό για την διαχείριση της επικοινωνίας μεταξύ view - controller και model - view.

- *Strategy:*
  Το Controller υλοποιείται συχνά με το Strategy ώστε να επιτρέπονται αλλαγές στους controllers.

5.2. Observers:

Έχουμε αναφέρει προηγουμένως πως το project μας προσφέρει έδαφος για περεταίρω ανάπτυξη και δημιουργία και καθώς πολλά παιχνίδια, στο διαδίκτυο ή όχι, διαθέτουν θέση παρατηρητή (**observer**) πιστεύουμε μπορεί να δημιουργηθεί ένα τέτοιος τύπος "παίκτη" όπου δε θα είναι ακριβώς παίκτης, αλλά απλός παρατηρητής, δηλαδή δε θα έχει κανένα δικαίωμα αλληλεπίδρασης με το grid ή οτιδήποτε άλλο γραφικό ή λειτουργία. Απλώς θα παρατηρεί το παιχνίδι από τη μεριά και των 2 παικτών. Στην ουσία θα παρατηρεί τα πάντα από την αρχή, θα γνωρίζει τις θέσεις των παικτών και θα βλέπει τα πάντα την ώρα που διεξάγεται η μάχη.

Για το λόγο αυτό, στην εκκίνηση του παιχνιδιού, δηλαδή στο σημείο λίγο πριν επιλέξουμε τύπο παίκτη, έχουμε παραθέσει ένα παράδειγμα του πως θα μπορούσε να μοιάζει η επιλογή observer (δηλαδή να επιλέξει κανείς να παρακολουθήσει ένα παιχνίδι που ήδη εξελίσσεται). Επίσης θα μπορούσαμε με κουμπιά για παράδειγμα να δηλώσουμε ζωντανά πόσους observers έχει μια εξελισσόμενη ναυμαχία. Αυτό θα μπορούσε να γίνει με **customized buttons**, όπου για κάθε button, και ένας παρατηρητής. Για του λόγου το αληθές, εμείς δείχνουμε με ένα παράδειγμα πως θα μπορούσε να μοιάζει.

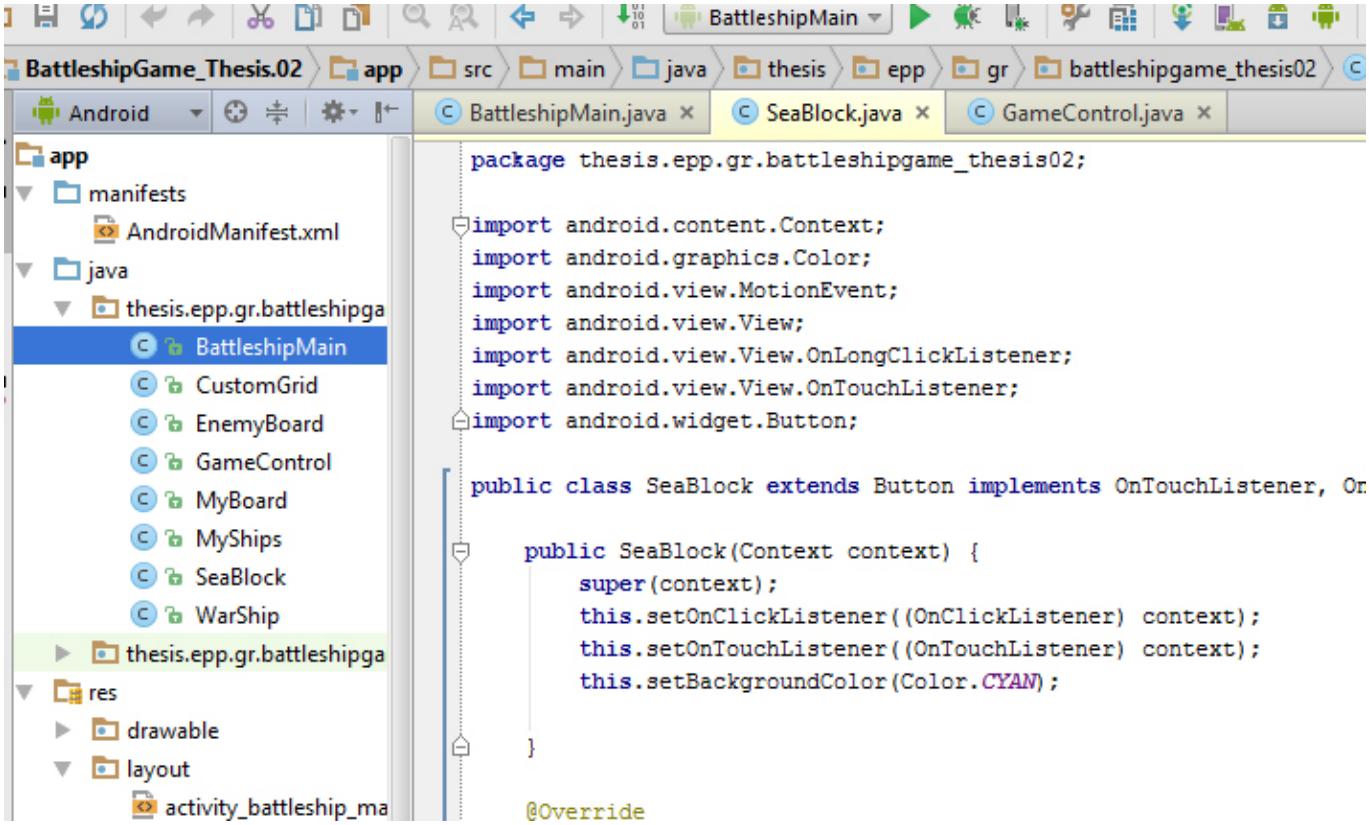
εικόνα 67 *- επιλογή ως observer*

5.3. Ανάπτυξη στο περιβάλλον Android:

Με την έκρηξη κινητικότητας στην ανάπτυξη λογισμικού σε περιβάλλον Android τα τελευταία χρόνια, θεωρούμε πως θα μπορούσε ο ίδιος ακριβώς κώδικας να μεταφραστεί σε Android περιβάλλον. Από την ιδέα της υλοποίησης του συγκεκριμένου παιχνιδιού ως και τον πολυμορφισμό που έχει επιτευχθεί, ακόμα και το δύσκολο κομμάτι MVC, ή το κομμάτι του TCP θεωρούμε καλή ιδέα και την επέκταση και στο συγκεκριμένο περιβάλλον. Μεγάλη γκάμα χρηστών θα μπορούσε να το προτιμήσει. Η γλώσσα παραμένει η ίδια, πρόκειται για αντικειμενοστραφή προγραμματισμό, σε διαφορετικό περιβάλλον εκτέλεσης και ανάπτυξης (**android mobile**).

Δοκιμάσαμε μερικά παρθενικά βήματα με τον IDE για Android Development: **Android Studio 1.0.1**. Πρόκειται για ένα πολύ σύγχρονο και πολύ λεπτομερές εργαλείο, που κάνει θαύματα και σε χέρια κάποιου αρχάριου. Αν κάποιος έχει καλή επαφή με το web search μπορεί να αναπτύξει απλό λογισμικό από την πρώτη κιόλας ημέρα και να το δει να τρέχει στη συσκευή του.

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389



εικόνα 68 *- περιβάλλον Android Studio*

Παρακάτω παρουσιάζουμε μερικές εικόνες με τους πειραματισμούς μας με τους listeners και πολλά άλλα εργαλεία. Το περιβάλλον, έχοντας δουλέψει σε NetBeans εμείς, φάνηκε εξαιρετικά οικείο. Απεικονίζουμε και μια μέθοδο, την getBlockPosition(), όπου είχαμε με τον ίδιο τρόπο ακριβώς και δομή αναπτύξει στο NetBeans.

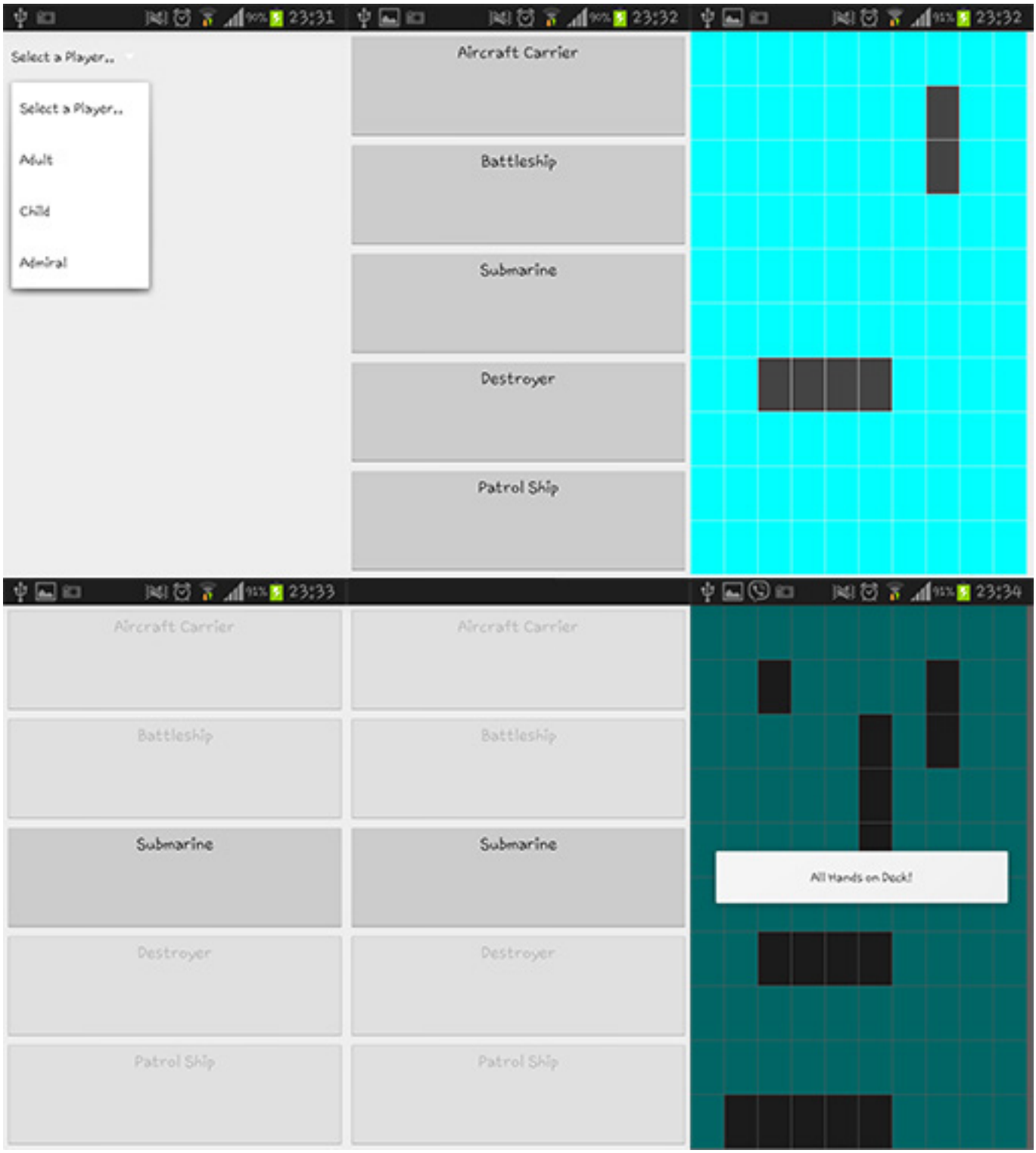
```
@Override
public boolean onTouch(View pressedView, MotionEvent event) {
    if (!seaBattleOn) {
        switch (event.getAction()) {
            case (MotionEvent.ACTION_DOWN):
                getBlockPosition(pressedView);
                x1 = event.getX();
                y1 = event.getY();
                break;
            case (MotionEvent.ACTION_UP):
                x2 = event.getX();
                y2 = event.getY();
                dx = x2 - x1;
                dy = y2 - y1;
                if (Math.abs(dx) > Math.abs(dy)) {
                    if (dx > 0) {
                        // RIGHT
                        if (coords[1] < (columns - (shipBlocks - 1))) {
                            if (checkCollision(3)) {
                                battleStations(3);
                                setContentView(myShips);
                            } else {
                                displayAlert(1);
                                myBoard.invalidate();
                            }
                        } else {
                            displayAlert(2);
                            myBoard.invalidate();
                        }
                    } else {
```

εικόνα 69 *- διαχειρίζοντας την OnTouch()*

```
/*
 * A method that gives each block unique coordinates.
 */
public void getBlockPosition(View pressedView) {
    View parentView = (ViewGroup) pressedView.getParent();
    for (int i = 0; i < ((ViewGroup) parentView).getChildCount(); i++) {
        if (((ViewGroup) parentView).getChildAt(i) == pressedView) {
            coords[0] = i / rows;
            coords[1] = i % columns;
            coords[2] = pressedView.getId();
            break;
        }
    }
}
```

εικόνα 70 *- μέθοδος getBlockPosition() σε Android*

Τσούτσας Γιώργος - 2542
Βίγκος Ιωάννης - 2389



εικόνα 71 *- αποτελέσματα και πειράματα στο mobile μας*

6. Βιβλιογραφία

6.1. Αναφορές & Internet sites:

[1] https://en.wikipedia.org/wiki/Battle-ship_%28game%29

[2] http://www.gametwist.gr/Arcade-%CE%BA%CE%B9-%CE%AC%CE%BB%CE%B-B%CE%B1/Battlefleet/%CE%9A%CE%B1%CE%BD%CF%8C%CE%B-D%CE%B5%CF%82%20%CF%80%CE%B1%CE%B9%CF%87%CE%B-D%CE%B9%CE%B4%CE%B9%CE%BF%CF%8D.html

[3] http://www.ebooks4greeks.gr/wp-content/uploads/2013/04/java-2012-eBooks-4Greeks.gr_.pdf

[4] http://www.ebooks4greeks.gr/dowloads/Pliroforiki/Glosses.program./Java__Down-loaded_from_eBooks4Greeks.gr.pdf

[5] https://netbeans.org/index_el.html

[6] http://artemis-new.cslab.ece.ntua.gr:8080/jspui/bitstream/123456789/5936/1/PD2008-0014.pdf

[7] http://www.computer.org/csdl/mags/so/2003/05/s5014.pdf

[8] http://www.medialab.ntua.gr/education/MultimediaTechnology/JavaNotes/files/8.%20Java_swing.pdf

[9] http://docs.oracle.com/javase/tutorial/uiswing/

[10] http://www.cise.ufl.edu/~amyles/tutorials/tcpchat/

[11] http://cgi.di.uoa.gr/~shadj/PLH36/tcp_chapter.pdf

[12] http://www.omg.org/mda/specs.htm#MDAGuide

[13] http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

6.2. Βιβλία:

Πλήρες εγχειρίδιο της Java 6:

- http://www.biblionet.gr/book/123696/Cadenhead,_Rogers/%CE%A0%CE%BB%CE%AE%CF%81%CE%B5%CF%82_%CE%B5%CE%B3%CF%87%CE%B5%CE%B9%CF%81%CE%AF%CE%B4%CE%B9%CE%BF_%CF%84%CE%B7%CF%82_Java_6