
**Πανεπιστήμιο Κρήτης –Τμήμα Επιστήμης Υπολογιστών**

**ΗΥ252– Αντικειμενοστρεφής Προγραμματισμός**

**Διδάσκων: Ι. Τζίτζικας**

**Χειμερινό Εξάμηνο 2020-2021**

# STRATEGO

# ICE VS FIRE

*Εισαγωγή*

Σπυρίδων Χρυσοβαλάντης Ζερβός

Csd4878

21/11/2022

# Περιεχόμενα

# 1. Εισαγωγή

Η εργασία χρησιμοποιεί το μοντέλο MVC (Model-View-Controller). Το Model περιέχει τα βασικά αντικείμενα του παιχνιδιού, αυτά δηλαδή με τα οποία τρέχει το παιχνίδι αφού περιέχουν το σημαντικότερο μέρος της πληροφορίας κατάστασης. Το View περιέχει όλες τις κλάσεις που αφορούν την διεπαφή με τους χρήστες (GUI), δηλαδή τον τρόπο με τον οποίο ο χρήστης αλληλοεπιδρά με τα αντικείμενα του. Τέλος, το Controller είναι η κλάση-διαιτητής του παιχνιδιού, αυτή δηλαδή που ελέγχει την ροή του προγράμματος και συγχρονίζει το Model με το View. Στις επόμενες τρείς ενότητες της αναφοράς γίνεται η ανάλυση των μερών του MVC. Συμπεριλαμβάνονται UML διαγράμματα για τις κλάσεις που συμμετέχουν σε κάποια ιεραρχία (π.χ. Piece). Οι βασικές λειτουργίες και οι αλγόριθμοι επεξηγούνται στην ενότητα 5, στην οποία περιλαμβάνονται και τα διαγράμματα UML του συνολικού προγράμματος. Η λειτουργικότητα και οι δυνατότητες του παιχνιδιού επεξηγούνται στην ενότητα 6 και, τέλος, στην ενότητα 7 δίνονται κάποια συμπεράσματα από την υλοποίηση του παιχνιδιού.

Σημείωση:

Στην αναφορά περιέχονται πολλά fields που, ενώ είναι final στον κώδικα, εδώ δεν δηλώνονται έτσι. Όσα πεδία δηλώνονται final εδώ αποτελούν σταθερές (const/define τύπου C). Τα υπόλοιπα final που είναι στον κώδικα άλλα όχι εδώ, δεν θεωρούνται σταθερές αλλά χρησιμοποιούν το final για λόγους ασφαλείας και εξαλείψεως των warnings.

# 2. Η Σχεδίαση και οι Κλάσεις του Πακέτου Model

Ανάλυση του Model:

- Package piece
  - Package movable:
    - Classes Beast Rider, Dragon, Elf, Knight, Lava Beast, Mage, Sorceress, Yeti

      Οι κλάσεις αυτές αντιπροσωπεύουν τα πιόνια του παιχνιδιού που δεν έχουν ειδικές δυνάμεις. Περιέχουν τα inherited από την MovablePiece fields και methods.

    - Classes Elf, Scout, Slayer

      Οι κλάσεις αυτές αντιπροσωπεύουν τα πιόνια του παιχνιδιού που έχουν ειδικές δυνάμεις. Περιέχουν τα inherited από την MovablePiece fields και methods, ενώ κάνουν implement το SpecialPiece. Το interface δεν προσφέρει κάτι παραπάνω όσον αφορά την υλοποίηση: Τα διαχωρίζει μόνο από τα υπόλοιπα pieces.

    - Abstract Class MovablePiece

      Όλα τα πιόνια που μπορούν να κινηθούν κάνουν extend αυτή την κλάση.

      Extended by classes: Beast Rider, Dragon, Elf, Knight, Lava Beast, Mage, Sorceress, Yeti, Elf, Scout and Slayer.

| **Field** | **Description** |
|---|---|
| Private int power | Power of the piece |
| Private boolean hasRescued | Whether the piece has rescued an other piece |

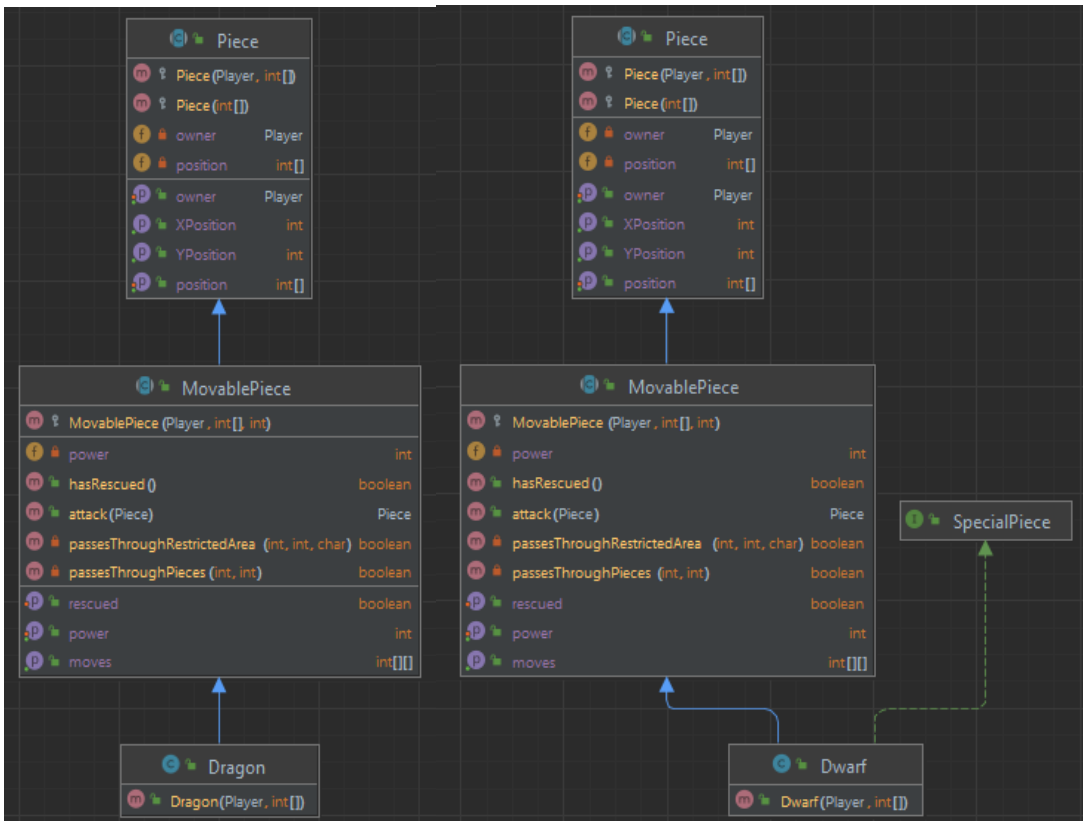
| **Method** | **Type** | **Description** |
|---|---|---|
| Public void setPower(int power); | Transformer (Mutative) | Sets the power of this piece |
| Public int getPower(); | Accessor | Gets the power of this piece |
| Public boolean hasRescued(); | Observer | Checks if the piece has rescued an other piece |

| | | |
|---|---|---|
| Public void setRescued(); | Transformer (Mutative) | Sets the piece rescue status |
| Public int[] getMoves(); | Accessor | Returns a list of available moves based on piece's current position on the board |
| Public Piece attack(Piece opp); | Transformer (Mutative) | Returns the piece that wins the fight while deleting the losing one |
| Private boolean passesThroughRestrictedArea(<br><br>int y, int x, char coord); | Observer | Checks if the piece, in order to go to position (y,x), has to pass through a restricted area. |
| Private boolean passesThroughPieces(int y, int x); | Observer | Checks if the piece, in order to go to position (y,x), has to pass through other pieces. |

Αλλαγές από Α φάση:
Προστέθηκε το πεδίο hasRescued με τις αντίστοιχες μεθόδους που χρησιμοποιεί η Controller::makeMove() για τους ελέγχους διάσωσης.
Μεταφέρθηκε η μέθοδος rescue στην Controller για να χρησιμοποιηθεί «πιο γενικά».
Προστέθηκαν οι μέθοδοι passesThroughRestrictedArea() και passesThroughPieces() για το «φιλτράρισμα» των κινήσεων από την getMoves().

- Interface SpecialPiece

  Χρησιμοποιείται ως "tag" των ειδικών πιονιών. Δεν προσφέρει κάποια λειτουργικότητα.




Τα UML διαγράμματα των Dragon και Dwarf

- Package stationary
  - Class Barrier

    Το μοναδικό πιόνι στο παιχνίδι που δεν έχει owner και χρησιμοποιεί τον constructor της StationaryPiece και Piece που δέχεται μόνο position. Χρησιμοποιείται για την αναπαράσταση των restricted areas του board.

    <u>Αλλαγές από Α φάση:</u>

    Η Barrier προστέθηκε για την ευκολότερη αναγνώριση των απαγορευμένων περιοχών. Είναι ιδιαίτερα χρήσιμη στην GameBoard για τις περιοχές που κάνει κλικ ο παίκτης.

  - Class Flag

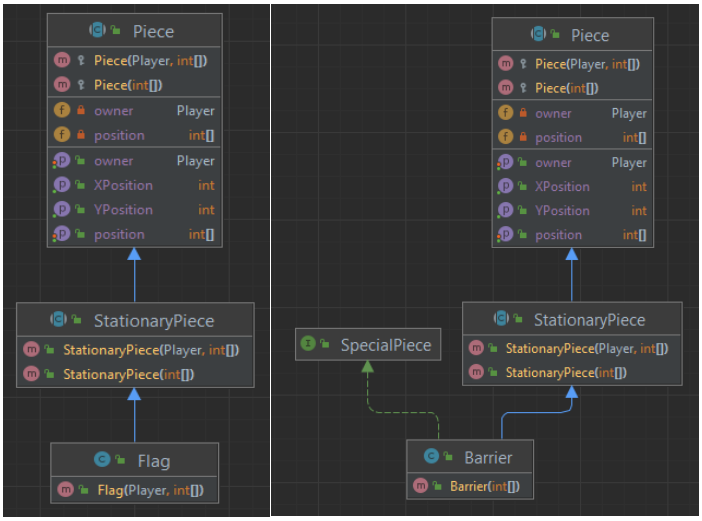
    Η κλάση αυτή αποτελεί τη σημαία του παίκτη. Πέραν της μεγάλης αξίας της για το παιχνίδι, περιέχει μόνον ό,τι κληρονομεί η StationaryPiece.

- Class Trap

  Το πιόνι-παγίδα που σκοτώνει ό,τι πέσει πάνω του, εκτός τον νάνο (dwarf). Περιέχει μόνον ό,τι κληρονομεί η StationaryPiece.

- Abstract Class StationaryPiece

  Extended by classes: Barrier, Flag, Trap.
  Χρησιμοποιείται για να διαχωρίσει τα ακίνητα pieces από τα κινούμενα. Περιέχει μόνο τα fields και τα methods της Piece.




Τα UML διαγράμματα της Flag και της Barrier

- Abstract Class Piece

  Η κεντρική κλάση που κάνει extend κάθε πιόνι. Οτιδήποτε την κάνει extend, θεωρείται πιόνι, κινούμενο ή όχι. Όπως αναφέρθηκε, κληρονομείται και από την Barrier η οποία όμως είναι ένα ιδιαίτερο πιόνι που δεν ανήκει σε κανέναν παίκτη και έχει ειδικό constructor για αυτήν.

| **Field** | **Description** |
|---|---|
| Private Player owner | Piece's owner – to which players the piece belongs to |
| Private int[] position | Its position on the board |

| Method | Type | Description |
|---|---|---|
| Public void setOwner(Player player); | Transformer (Mutative) | Sets the owner of the piece. |
| Public Player getOwner(); | Accessor | Gets the owner of the piece. |
| Public void setPosition(int[] position); | Transformer (Mutative) | Sets the position of the piece. |
| Public int[] getPosition(); | Accessor | Gets the position of the piece. |
| Public int getYPosition(); | Accessor | Gets the Y position of the piece (height). |
| Public int getXPosition(); | Accessor | Gets the X position of the piece (width). |

- Package Player
  - Class Army

    Η κλάση αυτή περιέχει τον στρατό (δηλαδή τα πιόνια) του παίκτη και ό,τι λειτουργείες σχετίζονται με αυτόν.

| Field | Description |
|---|---|
| Public static final int PIECE_TYPES (=11) | All different piece types, excluding the flag. |
| Public static final int[] DEFAULT_PIECES (=[1, 4, 5, 2, 2, 2, 3, 2, 1, 1, 6]) | Default number of pieces of each type in an army with reducedArmy option set to false. |
| Private String Owner | Army's owner |
| Private int[] activePieces | Active (alive) pieces of each type |
| Private ArrayList<ArrayList<Piece>> pieces | All pieces of the army (every piece instance) |

| Method | Type | Description |
|---|---|---|
| Public Player getOwner(); | Accessor | Gets the army's owner |
| Public int getPieceCode(Piece piece); | Transformer (Applicable) | Gets the piece code of the given piece. Piece codes are used mostly to locate piece information in Army's arrays. |
| Public void removePiece(Piece piece); | Tranformer (Mutative) | Removes the piece from the army |
| Public void addPiece(Piece piece); | Tranformer (Mutative) | Adds the piece to the army |
| Public int getNumberOfPieces(int piece); | Accessor | Gets the number of active pieces of the given piece type in the army |
| Public boolean hasMovablePieces(); | Observer | Checks if the army has active movable pieces |
| Public Piece createPieceFromCode(int pieceCode); | Transformer (Applicative) | Creates a piece based on the piece code for the army's owner |
| Public void createArmy(); | Transformer (Mutative) | Creates the army based on the gamemode. |
| Public boolean hasAvailableRescues(); | Observer | Checks if there are pieces that can be rescued |
| Public int[] getAvailableRescues(); | Accessor | Gets a list of the number of pieces that can be rescued from each type. |
| Public ArrayList<ArrayList<Piece>> getArmyPieces(); | Accessor | Gets the list of pieces in the army |
| Public int getActiveArmyPieces(); | Accessor | Gets the number of active pieces the army has |

Αλλαγές από Α φάση:

Προστέθηκαν οι getters getArmyPieces() και getActiveArmyPieces για την λειτουργία της Board::setArmyOnBoard() με σκοπό την αποφυγή συνεχών κλήσεων των μεθόδων της Army.

- Class Player

  Η κλάση αυτή είναι ουσιαστικά ο παίκτης και περιέχει τα στοιχεία του, χρήσιμες πληροφορίες και τις μεθόδους που σχετίζονται με αυτόν

| **Field** | **Description** |
|---|---|
| Private String name | Player's name. Defaults to player+<Id> |
| Private int Id | Player's id |
| Private Army army | Player's army (their pieces) |
| Private boolean flag | The status of the player's flag. True = alive, false = dead |
| Private int[] saves | Saves performed by each piece type |
| Private int totalSaves | Total saves performed |
| Private int[] captures | Captured opponent pieces of each type |
| Private int totalCaptures | Total captures performed |
| Private int totalAttacks | Total attacks performed |
| Private int successfulAttacks | Total successful attacks performed |
| Private int totalMoves | Total moves performed |

| | | |
|---|---|---|
| Public Army getArmy(); | Accessor | Gets player's army |
| Public void setName(String name); | Transformer (Mutative) | Sets player's name |
| Public String getName(); | Accessor | Gets player's name |
| Public void setId(int id); | Transformer (Mutative) | Sets player's Id |
| Public int getId(); | Accessor | Gets player's Id |
| Public void setFlag(boolean status); | Transformer (Mutative) | Sets flag status |
| Public boolean hasFlag(); | Observer | Checks if the flag is alive |
| Public void setSavesOf(int piece, int saves); | Transformer (Mutative) | Sets saves performed by a piece type |

| | | |
|---|---|---|
| Public int getSavesOf(int piece); | Accessor | Gets saves performed by a piece type |
| Public void setTotalSaves(int saves); | Transformer (Mutative) | Sets total saves |
| Public int getTotalSaves(); | Accessor | Gets total saves |
| Public void setCapturesOf(int piece, int captures); | Transformer (Mutative) | Sets captures performed by a piece type |
| Public int getCapturesOf(int piece); | Accessor | Gets captures performed by a piece type |
| Public void setTotalCaptures(int captures); | Transformer (Mutative) | Sets total captures |
| Public int getTotalCaptures(); | Accessor | Gets total captures |
| Public void setTotalAttacks(int attacks); | Transformer (Mutative) | Sets total attacks |
| Public int getTotalAttacks(); | Accessor | Gets total attacks |
| Public void setSuccessfulAttacks(int successfulAttacks); | Transformer (Mutative) | Sets successful attacks |
| Public int getSuccessfulAttacks(); | Accessor | Gets successful attacks |
| Public void setTotalMoves(int moves); | Transformer (Mutative) | Sets total moves |
| Public int getTotalMoves(); | Accessor | Gets total moves |

Αλλαγές από Α φάση:

Διαγράφηκε η resetPlayer() η οποία τελικά δεν χρειάστηκε για την επανεκκίνηση του παιχνιδιού.

- Class Board

  Η κλάση αυτή αποτελεί το ταμπλό του παιχνιδιού, δηλαδή εκεί που τοποθετούνται και περιέχονται τα πιόνια των δύο παικτών.

| **Field** | **Description** |
|---|---|
| Public static final int BOARD_WIDTH (=10) | Board's width (X) |
| Public static final int BOARD_HEIGHT (=8) | Board's height (Y) |
| Private Piece[][] board | The actual board, containing both player's pieces. |

| **Method** | **Type** | **Description** |
|---|---|---|
| Public boolean isRestrictedArea(int y, int x); | Observer | Checks whether the given position is restricted for the pieces to move into |
| Public Pieces getPieceAt(int y, int x); | Accessor | Gets the piece located at the given position |
| Public void setPieceAt(int y, int x, Piece piece); | Transformer (Mutative) | Sets the piece located at the given position |
| Public void setArmyOnBoard(Player player; | Transformer (Mutative) | Places player's army on the board based on their Id |
| Public boolean setPieceOnBoardRandomly(Piece piece); | Transformer (Mutative) | Places a piece randomly on board based on the player it belongs to |
| Public boolean isCoordValid(int y, int x); | Observer | Checks if the given position is valid |

Αλλαγές από Α φάση:

Προστέθηκαν οι setPieceOnBoardRandomly() για την τυχαία τοποθέτηση πιονιού μετά από διάσωση εντός του στρατού του παίκτη και isCoordValid() για την συντόμευση των ελέγχων περί coords.

- Class Timer

  Η κλάση ρολόι που μετράει τη διάρκεια του παιχνιδιού. Αποτελεί ένα thread.

| **Field** | **Description** |
|---|---|
| Private int[] time | The current time. Index 0, 1 or 2 for hours, minutes and seconds respectively |

| **Method** | **Type** | **Description** |
|---|---|---|
| Public void setTime(int[] time); | Transformer (Mutative) | Sets the time |
| Public int[] getTime(); | Accessor | Gets the current time |
| Public String toString(); | Accessor | Gets a formatted/readable string of the current time |
| Public void run(); | Transformer (Mutative) | Starts the clock |
| Public void updateTime(); | Transformer (Mutative) | Updates the time every second |
| Private int[] reformatTime (int[] time); | Transformer (Applicative) | Fixes invalid time representation |

Αλλαγές από Α φάση:

Προστέθηκε η reformatTime() για την διαχείριση και διόρθωση των μονάδων χρόνου.

Η κλάση Piece χωρίζεται αρχικά με βάση το αν το πιόνι μπορεί να κινηθεί και στη συνέχεια, για τα κινούμενα πιόνια, χρησιμοποιείται το interface SpecialPiece για να τα διαχωρίσει από τα υπόλοιπα, κάτι σαν tag.

Τα Movable Pieces έχουν όλα τα fields και methods που χρειάζονται για να χρησιμοποιηθούν. Η κίνηση διαχειρίζεται από την Controller. Παρόλα αυτά, η MovablePiece περιέχει τις μεθόδους για την επίθεση και την παροχή κινήσεων που μπορεί να πραγματοποιήσει το πιόνι. Συγκεκριμένα, με την attack το παρόν πιόνι επιτίθεται σε ένα αντίπαλο πιόνι και επιστρέφεται ο νικητής, τον οποίο χρησιμοποιεί στη συνέχεια η Controller. Η getMoves επιστρέφει έναν πίνακα από τοποθεσίες στις οποίες μπορεί το πιόνι να πάει από την παρούσα θέση του (Εδώ φαίνεται η ανάγκη χρήσης, σε κάποιες περιπτώσεις, της αναπαράστασης των τοποθεσιών με int[]).

Τα StationaryPieces δεν χρειάζονται κάτι παραπάνω αφού αναγνωρίζονται εντός της attack των MovablePieces.

Ο Player περιέχει ό,τι έχει να κάνει με τον παίκτη, δηλαδή στατιστικά, τον στρατό του, όνομα, την κατάσταση της σημαίας του κτλ. Η Army είναι όμως αυτή που διαχειρίζεται τον στρατό (δηλαδή τα πιόνια). Τα piece codes χρησιμοποιούνται για την άμεση χρήση των arrays που περιέχουν τα statuses των πιονιών και άλλες παρόμοιες πληροφορίες που σχετίζονται με τον τύπο του πιονιού και όχι με κάποιο συγκεκριμένο πιόνι. Εξάλλου, τα πιόνια, σε αντίθεση με τους παίκτες, δεν έχουν id. Έτσι, το μόνο που τα διαχωρίζει είναι ο owner και το position π.χ. αν κάποιο πιόνι γίνει rescue, θα δημιουργηθεί ένα νέο instance του και όχι το ίδιο που είχε γίνει capture. Αν και περιέχονται ως στατιστικά στην Player, η Army είναι η κλάση που δίνει τις απαραίτητες μεθόδους για τις διαθέσιμες διασώσεις (π.χ. hasAvailableRescues()) και της δημιουργίας/διαγραφής πιονιού για/από τον στρατό.

Η Board είναι ουσιαστικά το ταμπλό. Περιέχει τα πιόνια και τις μεθόδους που σχετίζονται με τις θέσεις του board. Οι θέσεις αριθμούνται από το 0 μέχρι το 7 για το ύψος και το 9 για το μήκος για να είναι πιο εύκολη η χρήση των positions εντός του κώδικα. Γι' αυτό το λόγο, όταν περνιέται ένα position σε μια μέθοδο δίνεται πρώτα το ύψος και μετά το μήκος π.χ. pos(5, 8) == pos(ύψος=5, μήκος=8), παρόλα αυτά, κάποιες μέθοδοι χρησιμοποιούν τον τρόπο (int y, int x) για συντομία. Εφόσον το παιχνίδι έχει ένα board, για την απλότητα της υλοποίησης, η κλάση board περιέχει μόνον στατικές μεθόδους. Περιέχει μεθόδους για την αναγνώριση τοποθεσιών ως επιτρεπτών ή μη (π.χ. isCoordValid()) και για την τοποθέτηση των στρατών σε αυτές (π.χ. setArmyOnBoard()) ή και ενός πιονιού που προήλθε από κάποια διάσωση (π.χ. setPieceOnBoardRandomly()).

Η Timer είναι απλώς ένα thread που μετράει τον χρόνο κατά τη διάρκεια του παιχνιδιού. Ξεκινάει να μετράει από την στιγμή που ξεκινάει το παιχνίδι και σταματάει μόλις νικήσει κάποιος παίκτης. Περιέχει μέθοδο toString για την αναπαράσταση του χρόνου καθώς και ό,τι χρειάζεται για την διαχείριση αυτού.

# 3. Η Σχεδίαση και οι Κλάσεις του Πακέτου Controller

| **Field** | **Description** |
|---|---|
| Private static Board board; | Game's board |
| Private static Player player1 | Player 1 (blue) |
| Private static Player player2 | Player 2 (red) |
| Private static Player nowPlaying | Player currently playing |
| Private static int round | Current round |
| Private static boolean reducedArmy | Reduced Army setting (true=enabled, false=disabled) |
| Private static boolean noRetreat | No Retreat setting (true=enabled, false=disabled) |
| Private static Timer timer | Timer (clock) of the game |
| Private static GameBoard gameboard | GameBoard View instance |
| Private static Statistics statistics | Statistics View instance |

| **Method** | **Type** | **Description** |
|---|---|---|
| Public static Board getBoard(); | Accessor | Gets board's instance |
| Public static Player getPlayer1(); | Accessor | Gets player 1's instance |
| Public static Player getPlayer2(); | Accessor | Gets player 2's instance |
| Public static Player getNowPlaying(); | Accessor | Gets player currently playing |
| Public static int getRound(); | Accessor | Gets current round number |
| Public static boolean getReducedArmy(); | Observer | Gets ReducedArmy setting |

| | | |
|---|---|---|
| Public static boolean getNoRetreat(); | Observer | Gets NoRetreat setting |
| Public static Timer getTimer(); | Accessor | Gets timer's instance |
| Public void startGame(String player1Name, String player2Name, boolean reducedArmyIn, Boolean noRetreatIn); | Transformer (Mutative) | Starts the game with the given options |
| Public static void endGame(Player winner); | Transformer (Applicative) | Ends the game with the given winner |
| Public static boolean checkIfEnded(); | Observer | Checks if the game has ended (if one of the two players has lost) |
| Public static void setPlayerPlaying(Player player); | Transformer (Mutative) | Changes the player playing |
| Public static void nextRound(); | Transformer (Mutative) | Moves on to the next round |
| Public static void makeMove(Piece piece, int[] position); | Transformer (Mutative) | Attempts to move a piece at a certain location. Calls any necessary actions if needed (e.g. attack, rescue). |
| Public static void initiateAttack(Piece attacker, Piece defender); | Transformer (Mutative) | Handles the attacking process between two pieces |
| Public static void rescue(Player player, int pieceCode); | Transformer (Mutative) | Handles the rescuing process of the given player's piece |
| Public static void main(String[] args); | Transformer (Mutative) | Initiates the game. |

Αλλαγές από Α φάση:

Προστέθηκε το field Statistics με τον αντίστοιχο getter για την αποθήκευση του Statistics View που λειτουργεί ως thread.

Προστέθηκαν οι μέθοδοι:

- rescue() που μεταφέρθηκε από την MovablePiece.
- initiateAttack() για την διαχείριση της επίθεσης έτσι ώστε η MovablePiece:attack() να έχει μόνο ως μέλημα την επιστροφή του νικητή της μάχης και να μην πειράζει το board.

Η Controller περιέχει όλα τα πεδία και τις μεθόδους που χρειάζεται για να «ενορχηστρώσει» το παιχνίδι: Από πεδία στατιστικών και λειτουργικών πληροφοριών μέχρι μεθόδους που αλληλοεπιδρούν και με το Model και με το View.

Η main του Controller ξεκινά το πρόγραμμα δημιουργώντας το MainMenu του View όπου οι παίκτες βάζουν τα ονόματα τους και επιλέγουν ρυθμίσεις. Το MainMenu καλεί (μόλις πατηθεί το play) την startGame με τις παραμέτρους που έδωσαν οι παίκτες, η οποία κάνει set το παιχνίδι και μπαίνει στον πρώτο γύρο. Κάθε φορά που γίνεται μια κίνηση καλείται η makeMove με το αντίστοιχο πιόνι και την κίνηση που πήγε να κάνει. Η makeMove καλεί όποια άλλη μέθοδο χρειαστεί για την πραγματοποίηση της κίνησης (π.χ. attack, rescue) και μετά την nextRound που προχωράει το παιχνίδι στον επόμενο γύρο, αν δεν έχει λήξει από την checkIfEnded(), δηλαδή κάνει update το board του GameView (μέσω της GameView::updateBoard()) και δίνει την σειρά στον επόμενο παίκτη με την setPlayerPlaying.

Οι αξιοσημείωτοι αλγόριθμοι και λειτουργίες επεξηγούνται στην ενότητα 5.

# 4. Η Σχεδίαση και οι Κλάσεις του Πακέτου View

Περιέχονται 4 Views:

- MainMenu

  Είναι το αρχικό – πρώτο view στο οποίο οι παίκτες βάζουν τα ονόματα τους και επιλέγουν ρυθμίσεις (reducedArmy, noRetreat). Όταν είναι έτοιμοι πατούν το play για να ξεκινήσει το παιχνίδι.

| **Field** | **Description** |
|---|---|
| Private JTextField player1TextField | Input field for player 1's name |
| Private JTextField player2TextField | Input field for player 2's name |
| Private JButton reducedArmyButton | Reduced Army setting input button |
| Private JButton noRetreatButton | No Retreat setting input button |
| Private JButton playButton | Play button (starts game) |

| **Method** | **Type** | **Description** |
|---|---|---|
| Public void actionPerformed(ActionEvent e); | Transformer (Mutative) | Handles settings and play buttons clicking events |

Αλλαγές από Α φάση:

Τα JCheckBoxes reducedArmy και noRetreat άλλαξαν σε JButtons για την καλύτερη αλληλεπίδραση με τον χρήστη.

- GameBoard

  Αποτελεί το ταμπλό, εκεί δηλαδή που στήνονται τα πιόνια και παίζουν οι παίκτες. Περιέχει μεθόδους για την επιλογή πιονιού, την εμφάνιση κινήσεων, την μετακίνηση πιονιού, την ανανέωση του board κτλ. καθώς και μερικές άλλες βοηθητικές μεθόδους.

| Field | Description |
|---|---|
| Private PieceButton[][] boardLayout | The layout of the clickable pieces on the board. Basically, a copy of Piece[][] board but with PieceButtons. |
| Private GridLayout layout | Board's layout |
| Private PieceButton pieceSelected | The piece currently selecting and showing its available moves |
| Private int[][] movesShown | Moves (positions) currently being shown on the board |

| **Method** | **Type** | **Description** |
|---|---|---|
| Public void actionPerformed(ActionEvent e); | Transformer (Mutative) | Handles all clicks on board |
| Private void selectPiece(PieceButton piece) | Transformer (Mutative) | Selects a piece and shows its available moves |
| Private boolean hasPieceSelected(); | Observer | Checks if a movable piece is selected |
| Public PieceButton[][] convertBoard(Board board); | Transformer (Applicative) | Converts a board of Pieces to a board of PieceButtons |
| Public void updateBoardView(Board board); | Transformer (Mutative) | Updates the GameBoard to the given board |

| | | |
|---|---|---|
| Public void clearMoves(); | Transformer (Mutative) | Clears marked positions (available moves) from the board |

Αλλαγές από Α φάση:

Προστέθηκαν τα πεδία layout, pieceSelected και movesShown για την ενημέρωση του board και την αποθήκευση του επιλεγμένου πιονιού το οποίου η κινήσεις φαίνονται στο board και αποθηκεύονται στο movesShown.

Αντίστοιχα, προστέθηκαν οι μέθοδοι hasPieceSelected() και selectPiece() για την διαχείριση του επιλεγμένου πιονιού και των κινήσεων που εμφανίζονται στο board. Τέλος προστέθηκε η clearMoves() για την εκκαθάριση των κινήσεων αυτών.

- Statistics

  Είναι το παράθυρο με τα στατιστικά και τις πληροφορίες του παιχνιδιού. Αποτελεί ξεχωριστό παράθυρο από το ταμπλό. Περιέχει τα fields που χρειάζονται για την διαχείριση του και μεθόδους για την ενημέρωση των πληροφοριών.

| Field | Description |
|---|---|
| Private Player playerPlaying | The player currently playing |
| Private Timer timer | Game's timer |
| Private Thread timerThread | Timer's thread instance |

| **Method** | **Type** | **Description** |
|---|---|---|
| Public void updateStatistics(); | Transformer (Mutative) | Updates all statistics and round info |
| Public void updateTimer(); | Transformer (Mutative) | Updates the timer (clock) only |
| Private ImageIcon getIcon(int pieceCode); | Transformer (Applicative) | Creates an ImageIcon of the piece code |

Αλλαγές από Α φάση:
Προστέθηκαν τα παραπάνω πεδία για την αποθήκευση βασικών και χαρακτηριστικών πληροφοριών του παραθύρου καθώς το παράθυρο βασίζεται στον παίκτη που παίζει εκείνη τη στιγμή και το timer αποτελεί ένα thread.
Προστέθηκε η μέθοδος getIcon για την δημιουργία των εικόνων των πιονιών στην captures section.

- GameEnd

Είναι το τελευταίο παράθυρο του παιχνιδιού, αυτό δηλαδή που εμφανίζεται όταν τελειώσει το παιχνίδι δείχνοντας τον νικητή και τα τελικά statistics. Έχει ένα κουμπί exit που απλώς βγαίνει από το παιχνίδι και ένα replay που το ξαναρχίζει.

| **Field** | **Description** |
|---|---|
| Private JButton replayButton; | Play again button |
| Private JButton exitButton | Exit game button |

| **Method** | **Type** | **Description** |
|---|---|---|
| Public void actionPerformed(ActionEvent e); | Transformer (Mutative) | Handles the exit and replay button click event |

Αλλαγές από Α φάση:

Προστέθηκε το field replayButton για την λειτουργία επανεκκίνησης του παιχνιδιού.

- PieceButton

  Αποτελεί μια «επέκταση» του Piece. Στην πραγματικότητα είναι το Piece αλλά περιέχει και ένα button που το κάνει clickable. Χρησιμοποιούνται από το GameBoard.

| **Field** | **Description** |
|---|---|
| Private Piece piece | The piece it represents |
| Private JButton button | The contained button so that it is clickable |

| **Method** | **Type** | **Description** |
|---|---|---|
| Public void setPiece(Piece piece); | Transformer (Mutative) | Sets the piece represented |
| Public Piece getPiece(); | Accessor | Gets the piece represented |
| Public void setButton(JButton button); | Transformer (Mutative) | Sets the button represented |
| Public JButton getButton(); | Accessor | Gets the piece represented |
| Private JButton setIcon(); | Transformer (Mutative) | Sets the icon of the button represented |

Αλλαγές από Α φάση:

Προστέθηκαν οι setters και οι getters των fields αφού τελικά ο constructor μόνο δεν έφτανε… όπως και η μέθοδος setIcon για την τοποθέτηση της εικόνας του πιονιού.

Κάθε κλάση παραπάνω εξηγεί τον εαυτό της και ποιος είναι ο ρόλος της. Η MainMenu είναι το αρχικό παράθυρο πριν ξεκινήσει το παιχνίδι, οι GameBoard και Statistics συνυπάρχουν με την πρώτη να είναι το ταμπλό που παίζουν οι παίκτες και η δεύτερη αναγράφει τα στατιστικά του παιχνιδιού και χρήσιμες πληροφορίες. Τέλος η GameEnd εμφανίζεται στο τέλος του παιχνιδιού αναγράφοντας τον νικητή και τα τελικά στατιστικά. Η Statistics λειτουργεί παράλληλα με την GameBoard, αποτελεί δηλαδή ένα thread, το οποίο αρχίζει μόλις δημιουργηθεί η GameBoard και τελειώνει μόλις κλείσει η τελευταία, δηλαδή στην αρχή και στο τέλος του παιχνιδιού αντίστοιχα.

Αρκετά ιδιαίτερη κλάση είναι όμως η PieceButton η οποία είναι ο τρόπος που αποκτούν τα πιόνια την δυνατότητα να χρησιμοποιηθούν από τον παίκτη, να μπουν δηλαδή μέσα στο παιχνίδι. Η PieceButton[][] που εμφανίζεται στην GameBoard είναι ένα αντίγραφο του Piece[][] board, δηλαδή του ταμπλό που περιέχεται στον Controller. Γι'αυτό το λόγο, μετά από κάθε γύρο και, γενικώς, όποτε γίνεται κάποια αλλαγή στο board (π.χ. μετακίνηση πιονιού) καλείται η updateBoardView της GameBoard, ώστε να ενημερωθεί το ταμπλό που βλέπουν οι παίκτες.

# 5. Η Αλληλεπίδραση μεταξύ των κλάσεων – Διαγράμματα UML

**Επεξήγηση κύριων λειτουργιών:**

- **Κίνηση**

Όταν είναι η σειρά του παίκτη, τα πιόνια του αντιπάλου είναι κρυμμένα και τα δικά του φαίνονται. Ο παίκτης κάνει κλικ σε οποιοδήποτε σημείο πάνω στο board και

  - Αν δεν έχει επιλέξει κάποιο πιόνι (hasPieceSelected()) και δεν είναι κάποιο stationary ή barrier (ή air/null προφανώς) το button που πάτησε – είπαμε πώς όλα στο board αναπαρίστανται από ένα PieceButton – τότε καλείται η *getMoves()* του πιονιού η οποία επιστρέφει έναν πίνακα από valid positions που μπορεί το πιόνι να κινηθεί. Μετά, καλείται η selectPiece() που θέτει αυτό το πιόνι ως επιλεγμένο και «πρασινίζει» τις δυνατές κινήσεις πάνω στο board.
  - Αν έχει ήδη επιλεγεί κάποιο πιόνι, δηλαδή αν υπάρχουν «πρασινισμένα κουτάκια» στο board, τότε ο παίκτης:
    - Αν πατήσει ένα πράσινο κουτάκι, θα κληθεί η makeMove() της controller που θα διαχειριστεί την κίνηση.
    - Διαφορετικά, γίνονται clear τα κουτάκια και επαναλαμβάνεται η διαδικασία

Μόλις ολοκληρωθεί η διαχείριση της κίνησης (βλ. παρακάτω), συνεχίζει ο άλλος παίκτης.

MovablePiece::getMoves()

Ο αλγόριθμος εύρεσης των δυνατών κινήσεων έχει ως εξής:

Προστίθενται αρχικά σε δύο παράλληλες λίστες (Μια για y positions και μια για x positions) όλες οι θέσεις που θα μπορούσε το πιόνι να πάει «σταυρωτά», δηλαδή σαν ένα σταυρό με το πιόνι στο κέντρο. Αν πρόκειται για scout, τότε ο σταυρός αυτός είναι από την μία άκρη του board στην άλλη. Στη συνέχεια, με βάση διάφορες περιπτώσεις, διαγράφονται οι λανθασμένες θέσεις (π.χ. αν το position βρίσκεται πάνω σε άλλο συμμαχικό πιόνι, αν για να πάει στη θέση περνάει από restricted area (barrier) κτλ). Για τον Scout χρησιμοποιείται η ειδική συνάρτηση passesThroughPieces() για να αφαιρέσει σημεία που, για να πάει, πρέπει να περάσει πάνω από άλλα πιόνια ή πίσω από αντίπαλο πιόνι, καθώς δεν εφαρμόζεται σε αυτόν η απλή εκκαθάριση βημάτων μεγαλύτερου του ενός τετραγώνου. Αν είναι ενεργοποιημένη η επιλογή, διαγράφονται και οι θέσεις υποχώρησης. Μόλις ολοκληρωθεί αυτό το «φιλτράρισμα» (όλες οι περιπτώσεις βρίσκονται εντός του κώδικα), δημιουργείται και επιστρέφεται ο τελικός πίνακας των positions.

- **Επίθεση**

Η επίθεση ξεκινάει από την makeMove(): Αν το position που επέλεξε το πιόνι δεν είναι null (δηλαδή κένο, άρα απλώς αλλάζει το position του πιονιού, ενημερώνεται το board και, αν δεν προηγείται κάποια διάσωση, συνεχίζει ο άλλος παίκτης), τότε

  - Αν είναι flag, απλώς θέτει false το flag status του αντιπάλου και καλεί την nextRound() η οποία με την σειρά την θα καλέσει την gameEnd() για την λήξη του παιχνιδιού.
  - Διαφορετικά, καλείται η initiateAttack() που θέτει στην μεταβλητή της winner τον νικητή της μάχης που προήλθε από την *attack()*. Με βάση τον νικητή ενημερώνονται καταλλήλως τα στατιστικά των δύο παικτών.
    *Σημείωση: winner==null σημαίνει ισοπαλία. Οι παγίδες δεν μετράνε ως προς τα στατιστικά αιχμαλώτισης, δηλαδή αν γίνει capture μια παγίδα, δεν θα προσμετρήσει.*
    Επιστρέφεται ο winner στην makeMove() και παίρνει την θέση του ηττημένου. Τότε, γίνεται ο έλεγχος αν η νέα θέση του νικητή είναι στην πρώτη γραμμή του αντιπάλου για να κληθεί η rescue() για τη διαδικασία της διάσωσης.

MovablePiece::attack()

Η attack() κάνει τους εξής ελέγχους με την ακόλουθη σειρά:

- Ελέγχονται οι ειδικές περιπτώσεις, δηλαδή τα ειδικά πιόνια. Συγκεκριμένα, ελέγχεται αν πρόκειται για μάχη μεταξύ επιτιθέμενου Slayer και αμυνόμενου δράκου ή νάνου και παγίδας ή σημαίας.
- Σκοτώνει το επιτιθέμενο πιόνι αν επιτίθεται σε παγίδα (ο έλεγχος για dwarf έχει γίνει παραπάνω).
- Διαφορετικά, επιστρέφει το ως προς power δυνατότερο πιόνι ή null αν είναι ισοδύναμα.

- **Διάσωση**

Αν μετά την ολοκλήρωση της κίνησης το πιόνι βρίσκεται σε θέση διάσωσης τότε γίνονται οι έλεγχοι σχετικά με την δυνατότητα του πιονιού (hasAvailableRescues()) και του παίκτη να πραγματοποιήσουν διάσωση. Αν ικανοποιούνται οι συνθήκες, δημιουργείται ένα παράθυρο επιλογής (RescueSelector) για να διαλέξει ο παίκτης το πιόνι που θέλει να διασώσει από αυτά που προσέφερε η *getAvailableRescues()*. Μόλις ο παίκτης διαλέξει πιόνι, τότε αυτό δίνεται στην rescue(), μαζί με τον παίκτη, για να γίνει η διάσωση. Η rescue() καλεί την createPieceFromCode() για να δημιουργήσει το πιόνι με βάση τον κωδικό του πιονιού (αυτόν δηλαδή που πήρε ως παράμετρο) και τον παίκτη, το οποίο στη συνέχεια προσθέτει στον στρατό του (addPiece()) και τοποθετεί τυχαία στο board (*setPieceOnBoardRandomly()*) ενημερώνοντας τις αντίστοιχες πληροφορίες. Αν αποτύχει η setPieceOnBoardRandomly(), τότε ακυρώνεται η διαδικασία της διάσωσης.

Έτσι, ολοκληρώνεται μια κίνηση και, σε κάθε περίπτωση, καλείται η nextRound() για να ξεκινήσει τον επόμενο γύρο – δηλαδή να παίξει ο άλλος παίκτης – ή να λήξει το παιχνίδι.

Army::getAvailableRescues()

Η λειτουργία της getAvailableRescues() είναι αρκετά απλή: Επιστρέφει έναν πίνακα, όσα και τα movable pieces, όπου - όπως σχεδόν σε όλο το πρόγραμμα άλλωστε - κάθε κελί αντιστοιχεί σε ένα τύπο πιονιού. Έτσι, σε κάθε κελί εισάγεται η διαφορά των default type amount και των active pieces του αντίστοιχου είδους με βάση το game mode (ReducedArmy). Ομοίως αν είναι ενεργοποιημένο το ReducedArmy, με την διαφορά ότι χρησιμοποιείται το default/2. Αν default/2==0 τότε θεωρείται το 1 ως default.

Board::setPieceOnBoardRandomly()

Η μέθοδος αναγνωρίζει αρχικά ποιανού παίκτη το πιόνι θα τοποθετήσει στο ταμπλό. Μετά επιλέγει δύο τυχαίους αριθμούς (y, x) για την θέση που θα προσπαθήσει να τοποθετήσει το πιόνι. Οι αριθμοί βρίσκονται φυσικά εντός των ορίων του στρατού του παίκτη, δηλαδή πίσω από τα barriers. Η διαδικασία επαναλαμβάνεται μέχρι να βρεθεί κάποια κενή θέση. Αν φτάσει τον μέγιστο αριθμό προσπαθειών χωρίς να βρεθεί κενή θέση, επιστρέφει false.

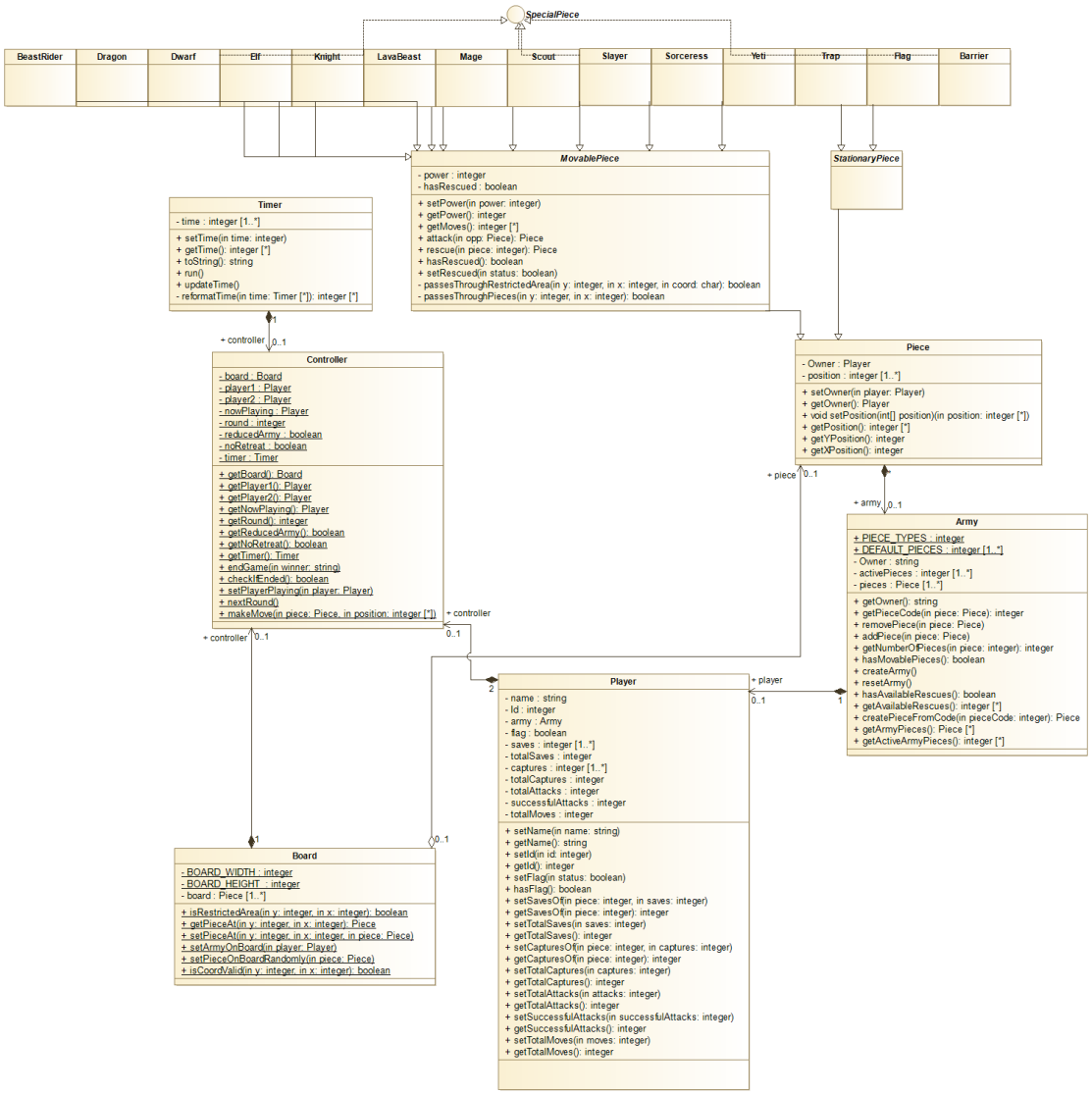
*Σημείωση: Όμοια σχεδόν λειτουργεί και η setArmyOnBoard(), αλλά για όλο το στρατό του παίκτη.*

Οι υπόλοιπες μέθοδοι περιέχουν σχόλια εντός του κώδικα. Εδώ επεξηγήθηκαν μόνον οι βασικές και πιο περίπλοκες που απαιτούνται για την κατανόηση των αλγορίθμων των λειτουργιών τριών αυτών βασικών λειτουργιών.

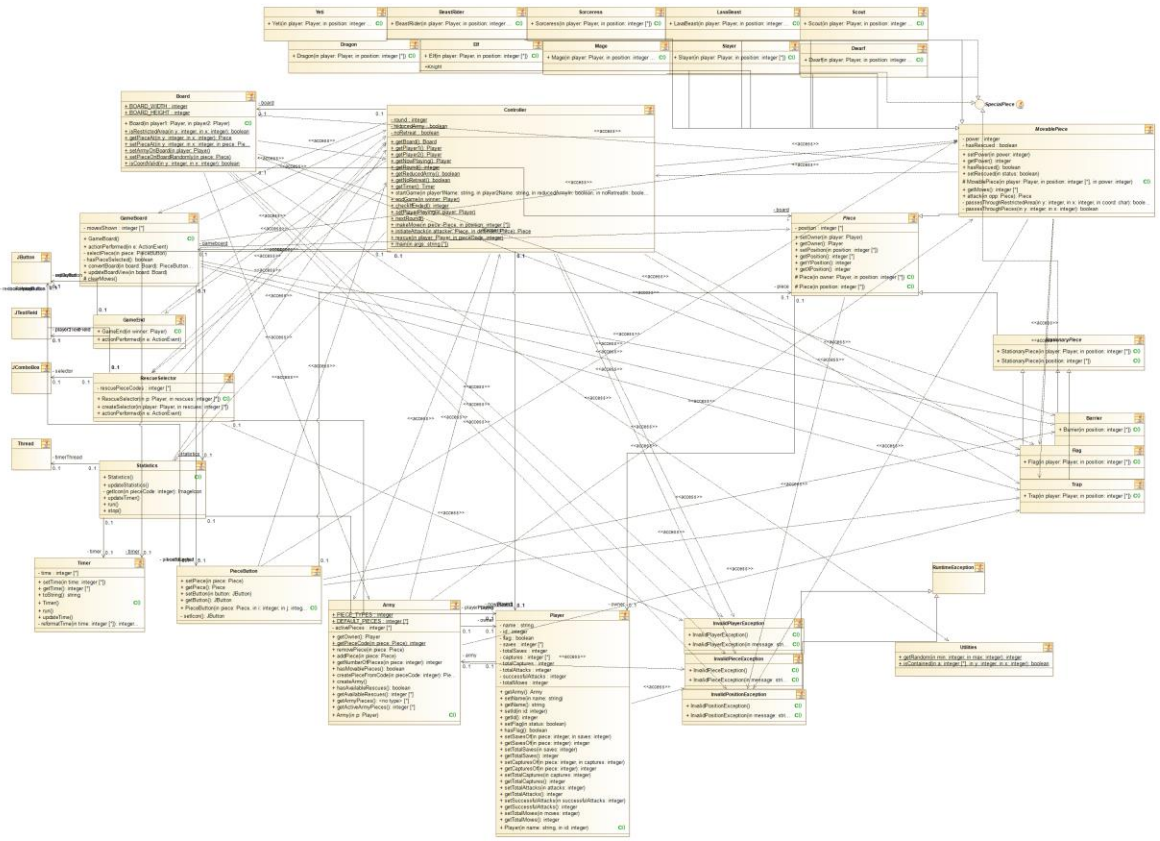
Συμπεριλαμβάνονται και μερικά JUnit Tests που ελέγχουν τις εξής λειτουργείες: Επίθεση, έλεγχος τοποθεσιών, διαχείριση γύρου/παίκτη, διάσωση και κίνηση.

Επιπλέον, στον κώδικα περιέχονται το πακέτο exception και utility. Το πρώτο περιέχει τα τρία exception InvalidPieceException, InvalidPlayerException και InvalidPositionException που αφορούν μη έγκυρα πιόνια, παίκτες και θέσεις αντίστοιχα. Και τα τρία αποτελούν Runtime Exceptions. Το δεύτερο περιέχει την κλάση Utility η οποία προσφέρει τις δύο βοηθητικές μεθόδους getRandom() και isContained() οι οποίες επιστρέφουν έναν τυχαίο αριθμό εντός κάποιων ορίων και ελέγχουν αν ένα ζευγάρι αριθμών υπάρχει στον δοσμένο πίνακα (πρόκειται για τη λίστα των positions και ένα position).

## Απλό/συνοπτικό UML διάγραμμα του παιχνιδιού



## Αναλυτικό/Ολοκληρωμένο διάγραμμα του παιχνιδιού



# 6. Λειτουργικότητα (Β Φάση)

Υλοποιήθηκαν όλα τα ζητούμενα της εργασίας, συμπεριλαμβανομένου του bonus NoRetreat game option.

# 7. Συμπεράσματα

Η εργασία ήταν αρκετά ενδιαφέρουσα, ειδικά στο κομμάτι του κώδικα της Β φάσης. Ομολογώ ότι η Α φάση ήταν αρκετά κουραστική σε σχέση με την Β καθώς περιείχε πολύ… «γραφειοκρατία» η οποία όμως, τελικά, βοήθησε σημαντικά κατά την υλοποίηση αφού προσέφερε ένα σημαντικό υπόβαθρο. Φυσικά υπήρξαν μικρές αλλαγές στην Β φάση (οι αλλαγές έχουν σημειωθεί στην αναφορά), όχι όμως πολλές. Οι σημαντικότερες έγιναν στο View, δηλαδή στα γραφικά. Τα διαγράμματα UML, είναι αρκετά χρήσιμα στην απλή μορφή τους (βλ. *Απλό/συνοπτικό UML διάγραμμα του παιχνιδιού*), όχι όμως όταν είναι πολύ αναλυτικά, όπως το δεύτερο UML διάγραμμα της ενότητας 5. Τα JUnit tests δεν θα έλεγα ότι βοήθησαν πολύ αφού μου φάνηκε ότι το project ήταν αρκετά μεγάλο για να χρησιμοποιηθούν. Παρόλα αυτά, περιέχονται μερικά στον κώδικα. Δεν υπάρχουν αξιοσημείωτες διαφοροποιήσεις σε σχέση με τους κανόνες που δόθηκαν ενώ το δυσκολότερο και με τα πιο πολλά προβλήματα κατά την υλοποίηση ήταν τα GameBoard και Statistics views όσον αφορά τα γραφικά.

Συνοψίζοντας, το πρότζεκτ ήταν αρκετά «διασκεδαστικό» αν και θα προτιμούσα να δίνονταν κάποιο άλλο μπόνους αντί του μπόνους παράδοσης (Παράδοση 23/12) πχ κάποια πιο περίπλοκη επιπρόσθετη λειτουργία.