Πανεπιστήμιο Κρήτης –Τμήμα Επιστήμης Υπολογιστών

HY252– Αντικειμενοστρεφής Προγραμματισμός

Διδάσκων: Ι. Τζίτζικας

Χειμερινό Εξάμηνο 2020-2021

AMPHIPOLIS PROJECT REPORT

Στέλιος Παπαμιχαήλ

A.M. 4020

25/11/2020

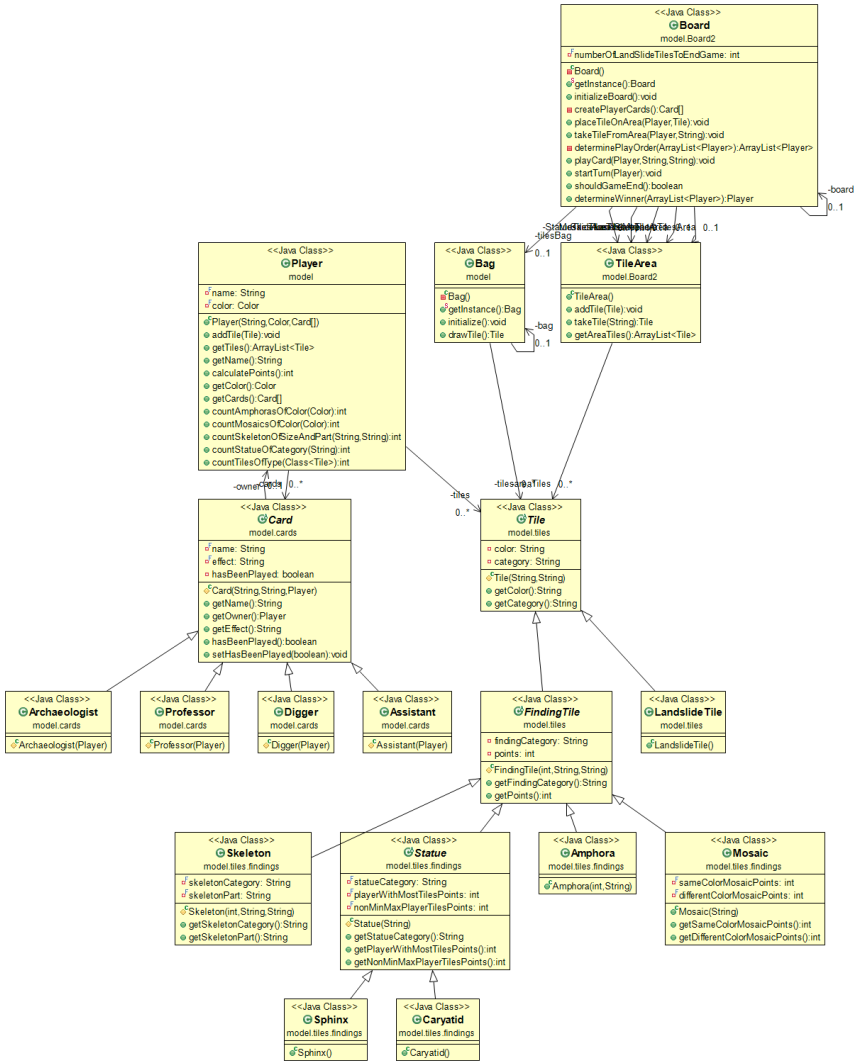
Περιεχόμενα

1. Εισαγωγή

Η εργασία της υλοποίησης του επιτραπέζιου παιχνιδιού Amphipolis θα βασιστεί στο μοντέλο αρχιτεκτονικής MVC (Model-View-Controller). Βάση αυτού, ο στόχος μας είναι η κλάση Controller να λειτουργήσει ως ο “¨εγκέφαλος” του παιχνιδιού, συνδέοντας και ελέγχοντας έτσι τα δεδομένα του παιχνιδιού, που βρίσκονται στο πακέτο Model, με την γραφική διεπαφή του, η οποία βρίσκεται στο πακέτο View. Στην συνέχεια της αναφοράς λοιπόν, θα αναλύσουμε τα πακέτα Model, View & Controller, καθώς και τις κλάσεις από τις οποίες απαρτίζονται.

2. Η Σχεδίαση και οι Κλάσεις του Πακέτου Model

Το πακέτο Model περιέχει όλες τις απαραίτητες κλάσεις έτσι ώστε να μοντελοποιήσουμε σωστά όλους τους τύπους δεδομένων που απαιτούνται για την σωστή αναπαράσταση των δεδομένων του παιχνιδιού. Αυτό συμπεριλαμβάνεις τις κλάσεις Tile (και τις υποκλάσεις της), Player, Bag, Board, TileArea, Card και τις υποκλάσεις της Card. Παρατίθεται συνοπτικά ένα UML Diagram του πακέτου με τις κλάσεις του.



2.1 Η κλάση Tile

Η κλάση Tile αναπαριστά ένα πλακίδιο του παιχνιδιού. Ορίζοντας την ως αφηρημένη, μας δίνεται η δυνατότητα να τοποθετήσουμε τα κοινά χαρακτηριστικά και λειτουργίες των διαφόρων πλακιδίων σε ένα μέρος. Κληρονομώντας την, οι διάφοροι τύποι πλακιδίων μπορούν να δηλώσουν επιπλέον και τα δικά τους ιδιαίτερα χαρακτηριστικά και λειτουργίες. Ακολουθούν τα χαρακτηριστικά και οι λειτουργίες της κλάσης Tile εξαιρουμένης της κατασκευάστριας μεθόδου η οποία αρχικοποιεί απλώς τα πεδία της κλάσης μιας και μια αφηρημένη κλάση δεν μπορεί να γίνει instantiated.

Attributes:

private String color; // The color of the tile

private String category; // The category of the tile

Methods:

public String getColor(); // Transformer : Returns the tile's color

public String getCategory(); // Transformer: Returns the tile's category

2.2 Οι υποκλάσεις της Tile

Τα πλακίδια του παιχνιδιού χωρίζονται σε δύο βασικές κατηγορίες, τα Landslide Tiles και τα Finding Tiles. Τα Landslide Tiles αποτελούν πραγματικά πλακίδια του παιχνιδιού ενώ τα Finding Tiles στην πραγματικότητα χωρίζονται σε πολλές υπο-κατηγορίες, όπως Αμφόρες,Μωσαϊκά,Σκελετούς και Αγάλματα. Για τον λόγο αυτό η κλάση FindingTile αποτελέι αφηρημένη κλάση σε αντίθεση με την κανονική κλάση LandslideTile.

- Η κλάση LandslideTile δεν έχει επιπλέον γνωρίσματα, και η μόνη μέθοδος της είναι η κατασκευάστρια της , η οποία απλά αρχικοποιεί τα πεδία της Tile μέσω της super().
- Ακολουθούν τα χαρακτηριστικά και οι λειτουργίες της κλάσης FindingTile:

Attributes:

private String findingCategory; // the category of the finding tile

private int points; // the points that the tile gives to the player

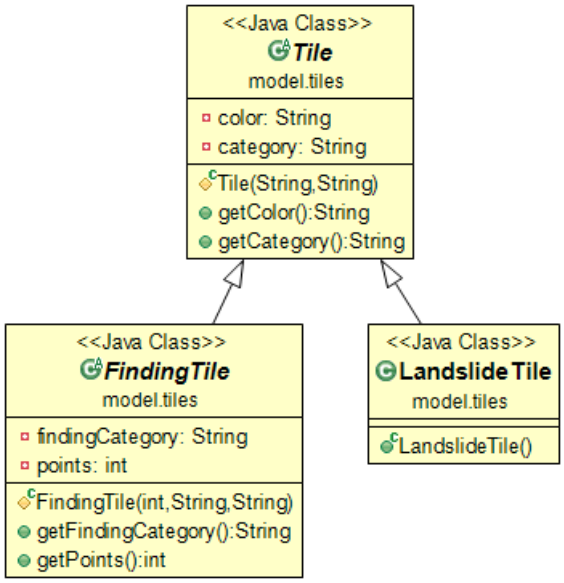
Methods:

protected FindingTile(int points, String color, String findingCategory); // Constructor

public String getFindingCategory(); // Accessor : Returns the category of the finding tile

public int getPoints(); // Accessor : Returns the points of the finding tile

Ακολουθεί ένα UML Diagram για τις κλάσεις Tile, FindingTile, LandslideTile.



2.3 Οι υποκλάσεις της FindingTile

Έχουμε τέσσερις υποκλάσεις, τις Amphora,Mosaic ,Skeleton & Statue. Η κλάση Amphora περιέχει μόνο την κατασκευάστρια μέθοδο για τις απαραίτητες αρχικοποιήσεις, η κλάση Mosaic έχει τα γνωρίσματα και τις μεθόδους που ακολουθούν:

Attributes:

private final int sameColorMosaicPoints; // The points to give when a player has 4 mosaic tiles of the same color

private final int differentColorMosaicPoints; // The points to give when a player has 4 mosaic tiles of different colors

Methods:

public Mosaic(String color); // Constructor

public int getSameColorMosaicPoints(); // Accessor: Returns the points for a four piece same color mosaic

public int getDifferentColorMosaicPoints(); // Accessor: Returns the points for a four piece different color mosaic

Ακολουθούν τα γνωρίσματα και οι μέθοδοι της κλάσης Skeleton:

Attributes:

private final String skeletonCategory; // The category of the skeleton tile. Either Big or Small

private final String skeletonPart; // The part of the skeleton tile. Either top or bottom

Methods:

public Skeleton(int points,String skeletonCategory, String skeletonPart); // Constructor

public String getSkeletonCategory(); // Accessor: Returns the skeleton's category

public String getSkeletonPart(); // Accessor: Returns the skeleton's part

2.3.1 Η κλάση Statue και οι υποκλάσεις της

Η κλάση Statue, έχει δηλωθεί ως αφαιρετική καθώς μπορεί να χωριστεί και αυτή σε υποκλάσεις, τις Sphinx & Caryatid. Ακολουθούν τα γνωρίσματα και οι μέθοδοι της κλάσης Statue.

Attributes:

private final String statueCategory; // The category of the statue, either Sphinx or Caryatid

private final int playerWithMostTilesPoints; // the points to give to the player with the most tiles

private final int nonMinMaxPlayerTilesPoints; // the points to give to the players that have neither the most nor the least amount of tiles

Methods:

protected Statue(String statueCategory); // "Constructor" used for initializations

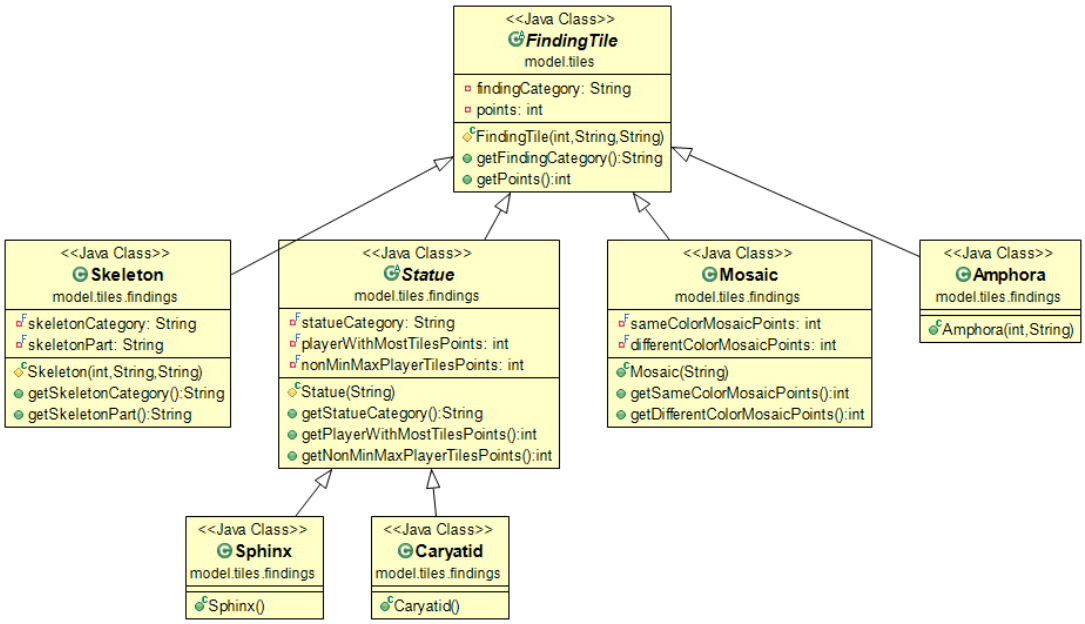
public String getStatueCategory(); // Accessor: Returns the statue's category

public int getPlayerWithMostTilePoints(); // Accessor: Returns the points to be given to the player with the most tiles

public int getNonMinMaxPlayerTilesPoints(); // Accessor: Returns the points to be given to the players with neither the most nor the least amount of tiles

Οι υποκλάσεις της κλάσης Statue, περιέχουν μόνο τις κατασκευάστριες μεθόδους τους για σωστή αρχικοποιήση των πεδίων της υπερκλάσης και δεν αναφέρονται παρακάτω.

- Ακολουθεί UML Diagram (FindingTile & subclasses)



2.4 Η κλάση Player

Η κλάση αυτή αναπαριστά έναν παίκτη του παιχνιδιού και διαχειρίζεται όλα τα απαραίτητα σχετικά δεδομένα, όπως η συλλογή πλακιδίων του, και προσφέρει τις επίσης απαραίτητες λειτουργίες επί αυτών. Ακολουθούν τα γνωρίσματα και οι μέθοδοι της κλάσης.

Attributes:

private final ArrayList<Tile> tiles; // Holds the player's tile collection

private final Card[] cards; // Holds the player's cards

private final String name; // The player's name

private final Color color; // The player's color

Methods:

public Player(String name, Color color, Card[] cards); // Constructor

public void addTile(Tile tile); // Transformer: adds the given tile to the player's collection

public ArrayList<Tile> getTiles(); // Accessor: returns the tile collection of the player's

public String getName(); // Accessor : returns the player's name

public int calculatePoints(); // Calculates and returns the points of the player based on his/her collection

public Color getColor(); // Accessor: returns the player's color

public Card[] getCards(); // Accessor : returns the player's card collection

public int countAmphorasOfColor(Color color); // Returns the number of amphoras with the given color in the player's tile collection

public int countMosaicOfColor(Color color); // Returns the number of mosaics with the given color in the player's tile collection

public int countSkeletonOfSizeAndPart(String size, String part); // Returns the number of skeleton tiles with the given size and part type in the player's tile collection

public int countStatueOfCategory(String category); // Returns the number of statue tiles with the given category in the player's tile collection

public int countTilesOfType(Class<? extends Tile> cls); // Returns the number of tiles of the given class in the player's tile collection. By using "? extends Tile" we specify that the function accepts only classes that are of type Tile. Those can only be subclasses of the Tile class since Tile is abstract. A more in depth explanation will be presented in phase B.

2.5 Η κλάση Bag

Η κλάση αυτή αναπαριστά την σακούλα με τα πλακίδια του παιχνιδιού και μας επιτρέπει να την γεμίσουμε με πλακίδια καθώς και να τραβάμε από αυτήν πλακίδια. Αξίζει να σημειωθεί εδώ πως η κλάση έχει υλοποιηθεί με τέτοιο τρόπο έτσι ώστε να είναι Singleton μιας και δεν μας χρειάζεται πάνω από μία οντότητα αυτής. Ακολουθούν τώρα τα γνωρίσματα και οι μεθόδοι της.

Attributes:

private final ArrayList<Tile> tiles; // The tiles of the bag

private static Bag bag; // used for storing the unique instance of the class

Methods:

private Bag(); // private constructor which prevents multi-instantiation

public static Bag getInstance(); // Accessor : returns the only instance of the class

public void initialize(); // Fills the bag with tiles so that we can draw from it

public Tile drawTile(); // Returns the drawn tile

2.6 Η κλάση Card και οι υποκλάσεις της

Η κλάση αυτή μοντελοποιεί μια κάρτα του παιχνιδιού. Αποτελεί αφαιρετική κλάση έτσι ώστε όλα τα κοινά γνωρίσματα και λειτουργίες των καρτών να βρίσκονται συγκεντρωμένα σε ένα μέρος. Οι διάφορες υποκλάσεις της (Assistant, Archaeologist, Digger & Professor), περιέχουν μόνο την κατασκευάστρια μέθοδό τους η οποία αρχικοποιεί απλώς τα κατάλληλα πεδία της Card ώστε η κάθε μια να έχει το δικό της, ξεχωριστό όνομα και ιδιότητα. Έτσι, παρακάτω ακολουθούν μόνο τα γνωρίσματα και οι λειτουργίες της Card.

Attributes:

private final String name; // the card's name

private final Player owner; // the owner of the card

private final String effect; // the effect of the card

private boolean hasBeenPlayed; // keeps track of whether or not the card has been played

Methods:

protected Card(String name, String effect, Player owner); // "Constructor" used to initialize the fields of the class

public String getName(); // Accessor: returns the card's name

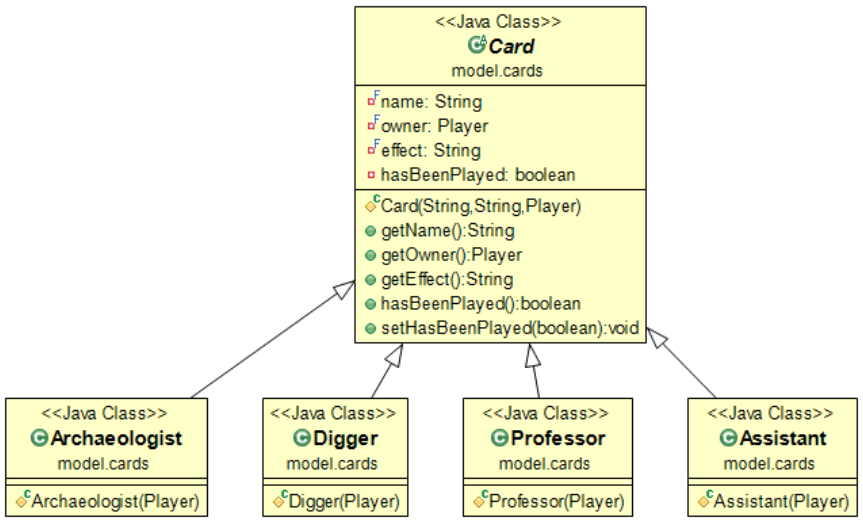
public String getEffect(); // Accessor: returns the card's effect

public Player getOwner(); // Accessor: returns the owner of the card

public boolean hasBeenPlayed(); // Observer: returns true if the card has been played or false otherwise

public void setHasBeenPlayed(boolean hasBeenPlayed); // Transformer: sets the hasBeenPlayed field to the given value

- Ακολουθεί ένα UML diagram για την κλάση Card και τις υποκλάσεις της.



2.7 Η κλάση TileArea

Αυτή η κλάση αναπαριστά μια περιοχή τοποθέτησης πλακιδίων στο ταμπλό. Παρακάτω παρατίθενται τα γνωρίσματα και οι μέθοδοι της κλάσης.

Attributes:

private final ArrayList<Tile> areaTiles; // The tiles the area will hold

Methods:

public TileArea(); // Constructor

public void addTile(Tile tile); // adds a tile to the tile area

public Tile takeTile(String tileCategory); // Returns a tile from the area that matches the given category

public ArrayList<Tile> getAreaTiles(); // Accessor: returns the tiles of the area

2.8 Η κλάση Board

Η τελευταία κλάση του πακέτου Model, είναι η κλάση Board, η οποία αναπαριστά το ταμπλό του παιχνιδιού. Αυτό περιέχει 5 περιοχές πλακιδίων όπου τοποθετούνται και αφαιρούνται πλακίδια κατά την διάρκεια του παιχνιδιού. Η κλάση αυτή περιέχει επίσης μεθόδους υπεύθυνες για τη σωστή αρχικοποίηση, διεξαγωγή και τερματισμό του παιχνιδιού, όπως η αρχικοποίηση της σακούλας πλακιδίων, η τυχαία επιλογή σειρών των παικτών, η τοποθέτηση πλακιδίων σε κάποια περιοχή, κλπ. Αξίζει επιπλέον να σημειωθεί πως, η κλάση αποτελεί Singleton εφόσον κάθε παιχνίδι απαιτεί ένα ταμπλό και μόνο κάθε φορά. Παρακάτω παρατίθενται τα γνωρίσματα καθώς και οι μέθοδοι της κλάσης.

Attributes:

private final Bag tileBag; // the tile bag to be used for the game

private final int numberOfLandSlideTilesToEndGame; // holds the number of landslide tiles needed on the board, for the game to end

private final TileArea landslideTileArea; // area holding landslide tiles

private final TileArea amphoraTileArea; // area holding amphora tiles

private final TileArea mosaicTileArea; // area holding mosaic tiles

private final TileArea statueTileArea; // area holding statue tiles

private final TileArea skeletonTileArea; // area holding skeleton tiles

private static Board board; // variable for accessing the only instance of the class

Methods:

private Board(); // private constructor to avoid multi-instantiation

public static Board getInstance(); // Accessor : returns the only instance of the class

public void initializeBoard(); // Performs all necessary initialization to guarantee that the game will flow as expected.

private Card[] createPlayerCards(); // Creates a set of cards for a player

public void placeTileOnArea(Player player, Tile tile); // places the given tile from the passed player's collection to the appropriate tile area.

public void takeTileFromArea(Player player, String tileCategory); // Removes a tile matching the given tile category from the appropriate area and adds it to the passed player's collection.

private ArrayList<Player> determinePlayOrder(ArrayList<Player> players); // Randomly decides the order the players will play in and returns a list of them in the order they will be playing

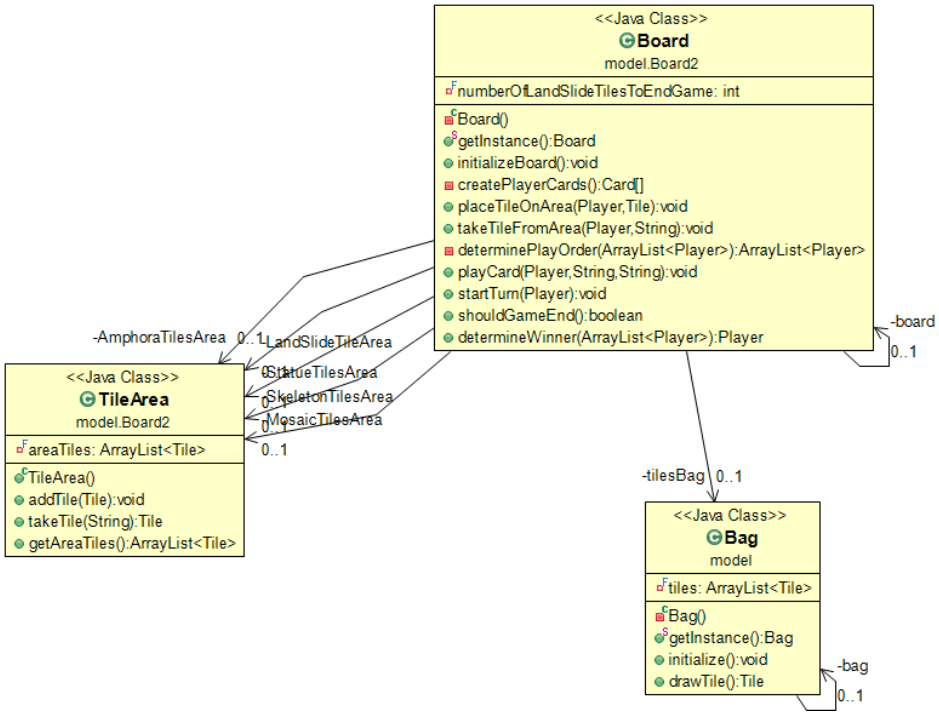
public void playCard(Player player, String cardName, String chosenArea); // Handles the logic when a player tries to play a card

public void startTurn(Player player); // Performs all actions necessary every turn for the given player

public boolean shouldGameEnd(); // Observer: returns true if the termination condition is satisfied or false otherwise

public Player determineWinner(ArrayList<Player> players); // Determines the winner of the game once it has ended

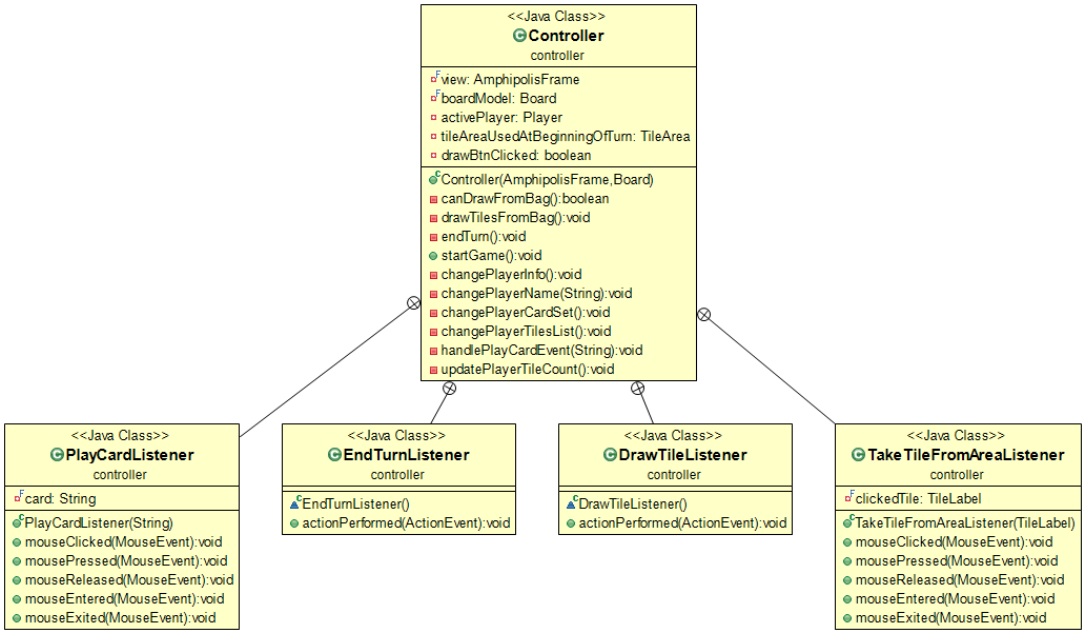
- Ακολουθεί ένα UML Diagram της κλάσης Board



3. Η Σχεδίαση και οι Κλάσεις του Πακέτου Controller

Το πακέτο Controller περιέχει μία μόνο κλάση, την Controller. Η κλάση αυτή είναι ο “εγκέφαλος” του παιχνιδιού, με την έννοια ότι είναι υπεύθυνη για την επικοινωνία και την διαχείριση του Model και του View έτσι ώστε το παιχνίδι να μπορεί να διεξαχθεί ομαλά και σωστά. Οι μέθοδοι της κλάσης αυτής, χρησιμοποιούνται έτσι ώστε κάθε αλληλεπίδραση του χρήστη με το View να αντικατοπτρίζεται και στα δεδομένα του παιχνιδιού (Model), αλλά και το αντίστροφο. Δηλαδή, μια αλλαγή των δεδομένων του παιχνιδιού στο Model, θα επιφέρει αλλαγή και στο View του παιχνιδιού.

Όπως θα δούμε παρακάτω μέσω των γνωρισμάτων και του UML diagram, η Controller έχει μεταξύ άλλων, ένα πεδίο για το view, και ένα πεδίο για το Model. Μέσω αυτών, μπορεί να προσπελάσει όλα τα απαραίτητα δεδομένα και ενέργειες (μέσω της διεπαφής που οι κλάσεις αυτές ορίζουν), για να εγγυηθεί την ομαλή διεξαγωγή του παιχνιδιού. Ακολουθεί το διάγραμμα UML του πακέτου. Τα επιμέρους τμήματα του αναλύονται παρακάτω.



3.1.1 Η κλάση Controller

Εδώ παρουσιάζονται τα γνωρίσματα και οι μέθοδοι της κλάσης, εξαιρουμένων των εσωτερικών της κλάσεων, οι οποίες αναλύονται αμέσως μετά.

Attributes:

private final AmphipolisFrame view; // holds an instance of the GUI of the game thus allowing access to the various UI elements of the game

private final Board boardModel; // holds an instance of the board model of the game, thus allowing access to the game's data

private Player activePlayer; // keeps track of the player who's playing this turn

private TileArea tileAreaUsedAtBeginningOfTurn; // tracks the tile area that the active player took tiles from when his/her turn started. This is useful for when using Cards since some effects rely on this information to work.

private boolean drawBtnClicked; // keeps track of whether or not the draw button of the UI has been clicked. This is used to avoid cases where a player could draw tiles from the bag multiple times per round.

Methods:

public Controller(AmphipolisFrame view, Board boardModel); // Constructor of the class which initializes the view and model fields with the given parameters. These

parameters will be supplied by the main method which is the starting point of the game as we will see later on.

private boolean canDrawFromBag(); // Observer : returns true if the draw button has already been used this turn or false otherwise

private void drawTilesFromBag(); // This method is called when the draw button of the UI is clicked by a player

private void endTurn(); // This method is called when the End Turn button of the UI is clicked by a player

public void startGame(); // This method is responsible for showing the GUI of the game and starting the game

private void changePlayerInfo(); // This method is responsible for changing all player related info, such as Player name, card collection, tile collection , etc. when another player has ended his/her turn.

private void changePlayerName(); // Method used by changePlayerInfo(). Responsible for changing the active player's name in the UI when a new turn begins

private void changePlayerCardSet(); // Method used by changePlayerInfo(). Responsible for changing the active player's card collection in the UI when a new turn begins.

private void changePlayerTilesList(); // Method used by changePlayerInfo(). Responsible for changing the active player's tile collection in the UI when a new turn begins.

private void handlePlayCardEvent(String cardName); // Method responsible for handling the logic when a card from the player's collection is played

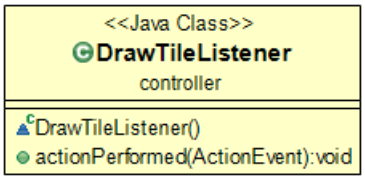
private void updatePlayerTileCount(); // This method is used to update the count of the player's tile collection. i.e. when the player takes a tile from an area, this function will increase the appropriate tile's count in the player's collection to reflect that event

3.1.2 Οι εσωτερικές κλάσεις της Controller

Μέσα στην κλάση Controller, ορίζονται και τέσσερις εσωτερικές κλάσεις , οι EndTurnListener, TakeTileFromAreaListener, PlayCardListener & DrawTileListener. Κάποιες από αυτές υλοποιούν την διεπαφή MouseListener, ενώ κάποιες άλλες την διεπαφή ActionListener. Αυτές οι εσωτερικές κλάσεις είναι χρήσιμες διότι επιτρέπουν στην Controller να ορίσει το τι πρέπει να γίνει όταν ο χρήστης αλληλεπιδρά με ορισμένα τμήματα του View, παραμένοντας έτσι ο εγκέφαλος του παιχνιδιού και διατηρώντας έτσι την αρχιτεκτονική MVC. Παρατίθενται τώρα οι εσωτερικές αυτές κλάσεις. Οι κλάσεις οι οποίες υλοποιούν την διεπαφή MouseListener , παρόλο που είναι αναγκασμένες να κάνουν override όλες τις μεθόδους της διεπαφής, εμείς εδώ χρησιμοποιούμε μόνο την mousePressed(MouseEvent e) για τον χειρισμό γεγονότων και για τον λόγο αυτό, μόνο αυτή θα συμπεριληφθεί παρακάτω στη λίστα μεθόδων των κλάσεων αυτών.

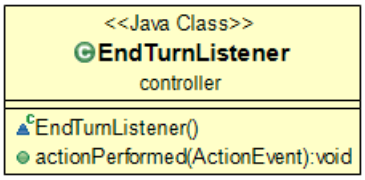
1. Η εσωτερική κλάση DrawTileListener

   Αυτή η κλάση είναι υπεύθυνη για τον χειρισμό της λογικής όταν ένας παίχτης επιλέγει να τραβήξει ένα πλακίδιο από την σακούλα μέσω της γραφικής διεπαφής. Υλοποιώντας την διεπαφή ActionListner , μπορούμε να την ορίσουμε σαν ActionListener του Draw Button της διεπαφής που θα δούμε στο πακέτο View για να χειρίζεται αυτό το γεγονός. Δίνεται το UML diagram της κλάσης.



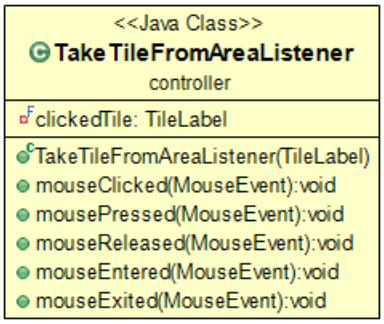
2. Η εσωτερική κλάση EndTurnListener

   Αυτή η κλάση είναι υπεύθυνη για τον χειρισμό της λογικής όταν ένας παίχτης επιλέγει να τελειώσει την σειρά του πατώντας το κουμπί End Turn της γραφικής διεπαφής. Υλοποιώντας την διεπαφή ActionListner , μπορούμε να την ορίσουμε σαν ActionListener του End Turn Button της διεπαφής που θα δούμε στο πακέτο View για να χειρίζεται αυτό το γεγονός. Δίνεται το UML diagram της κλάσης.



3. Η εσωτερική κλάση TakeTileFromAreaListener

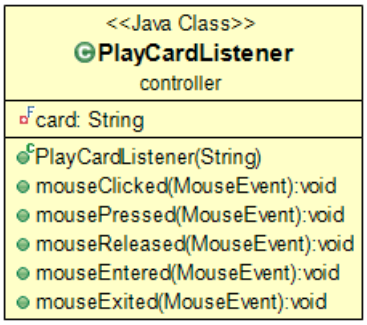
   Αυτή η κλάση είναι υπεύθυνη για τον χειρισμό της λογικής όταν ένας παίχτης επιλέγει να τραβήξει ένα πλακίδιο από κάποια περιοχή του ταμπλό , πατώντας πάνω σε αυτό το πλακίδιο της γραφικής διεπαφής. Υλοποιώντας την διεπαφή MouseListener , μπορούμε να την ορίσουμε σαν MouseListener των διαφόρων πλακιδίων της γραφικής διεπαφής για να χειρίζεται αυτό το γεγονός. Ο λόγος που εδώ χρειαζόμαστε MouseListener αντί για ActionListener, είναι διότι το κάθε πλακίδιο θα αναπαρασταθεί μέσω ενός Jlabel το οποίο είναι τύπου Component και έτσι δεν δέχεται ActionListeners. Βλέπουμε στο UML diagram της κλάσης ότι περιέχει επίσης και μια κατασκευάστρια μέθοδο, την TakeTileFromAreaListener(TileLabel tile), η οποία παίρνει ως όρισμα την TileLabel (κλάση του πακέτου View που θα δούμε αργότερα, αναπαριστά ένα πλακίδιο στην γραφική διεπαφή), έτσι ώστε να ξέρει ποιο TileLabel θα πρέπει να τροποποιήσει όταν πατηθεί από τον παίχτη.



4. Η εσωτερική κλάση PlayCardListener

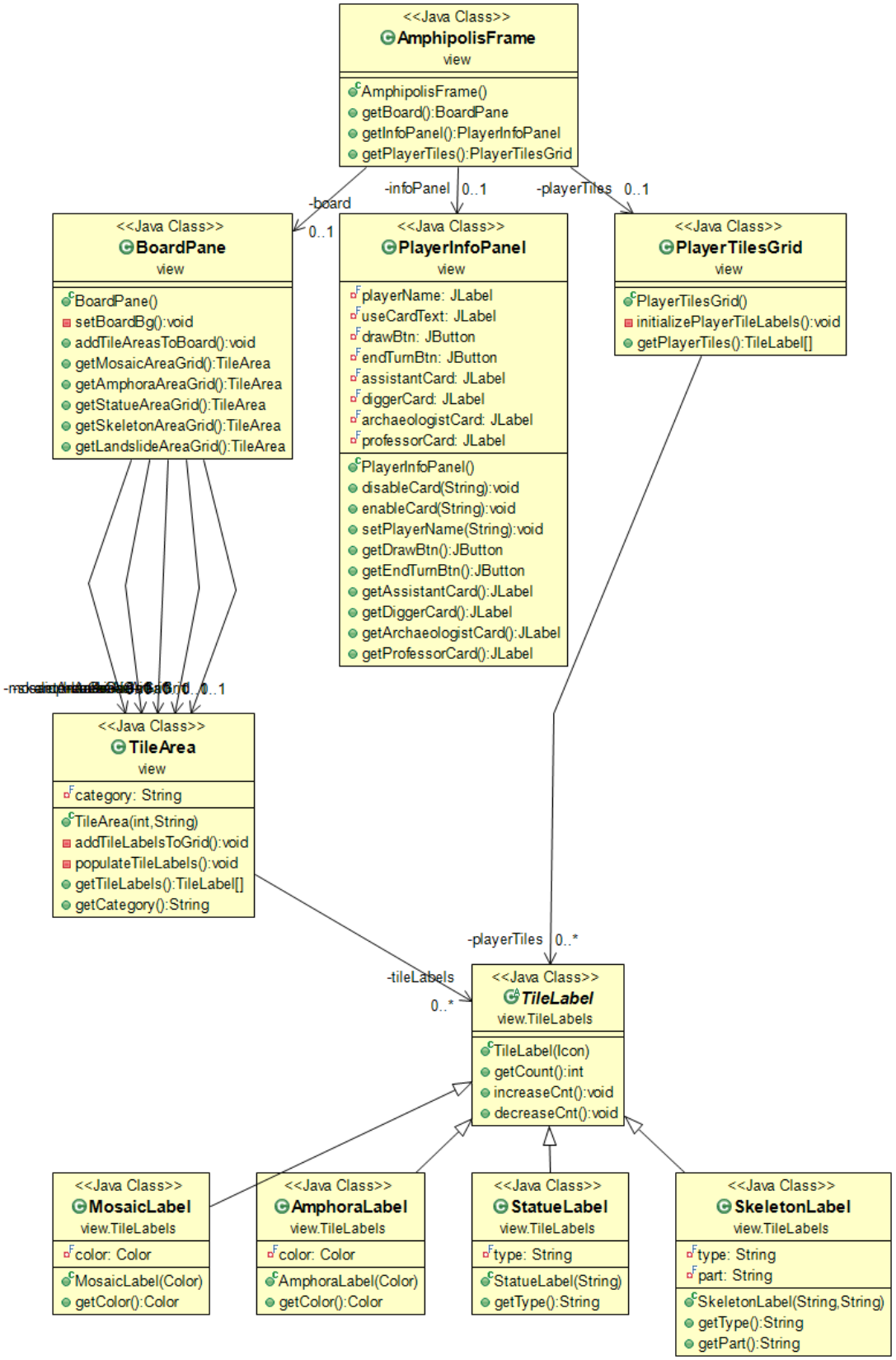
   Αυτή η κλάση είναι υπεύθυνη για τον χειρισμό της λογικής όταν ένας παίχτης επιλέγει να παίξει μια κάρτα, πατώντας πάνω σε αυτήν στη γραφική διεπαφή. Υλοποιώντας την διεπαφή MouseListener , μπορούμε να την ορίσουμε σαν MouseListener των διαφόρων καρτών της γραφικής διεπαφής για να χειρίζεται αυτό το γεγονός. Ο λόγος που εδώ χρειαζόμαστε MouseListener αντί για ActionListener, είναι πάλι διότι η κάθε κάρτα θα αναπαρασταθεί μέσω ενός Jlabel το οποίο είναι τύπου Component και έτσι δεν δέχεται ActionListeners. Βλέπουμε στο UML diagram της κλάσης ότι περιέχει επίσης και μια κατασκευάστρια μέθοδο, την PlayCardListener(String cardName), η οποία παίρνει ως όρισμα ένα String που θα αντιστοιχεί στην κάρτα που επέλεξε να παίξει ο παίχτης, έτσι

ώστε να ξέρει ποια ιδιότητα ποιας κάρτας θα πρέπει να ενεργοποιήσει (αν δεν έχει ήδη παιχτεί).



4. Η Σχεδίαση και οι Κλάσεις του Πακέτου View

Το πακέτο αυτό περιέχει όλες τις κλάσεις που αποτελούν την γραφική διεπαφή του παιχνιδιού. Κάθε μια αποτελεί υποκλάση κάποιας κλάσης της βιβλιοθήκης swing της Java με σκοπό την επέκταση των λειτουργιών τους όπως κρίνεται απαραίτητο για να μπορέσει να αναπαρασταθεί το παιχνίδι επαρκώς μέσω αυτής της γραφικής διεπαφής. Θα δούμε τώρα συνοπτικά μια σύντομη περιγραφή των διαφόρων κλάσεων του πακέτου. Μια σημαντική κλάση, είναι η αφηρημένη κλάση TileLabel η οποία αναπαριστά ένα πλακίδιο του παιχνιδιού και χωρίζεται σε υποκλάσεις βάση του τύπου του πλακιδίου. Στο πακέτο υπάρχουν επίσης οι κλάσεις, TileArea που αναπαριστά γραφικά μια περιοχή τοποθέτησης πλακιδίων στο ταμπλό, η PlayerTilesGrid, η οποία αναπαριστά την συλλογή πλακιδίων του εκάστοτε παίχτη, η PlayerInfoPanel , που αναπαριστά το τμήμα της διεπαφής που περιέχει τις επιπλέον πληροφορίες που σχετίζονται με έναν παίχτη (κάρτες, όνομα, κλπ.), η BoardPane που αντιστοιχεί στο ταμπλό του παιχνιδιού, και τέλος, η AmphipolisFrame η οποία συγκεντρώνει και συνδέει όλα τα παραπάνω σε ένα Jframe ώστε να υλοποιήσει το τελικό GUI του παιχνιδιού. Παρακάτω δίνεται το UML διάγραμμα του πακέτου.

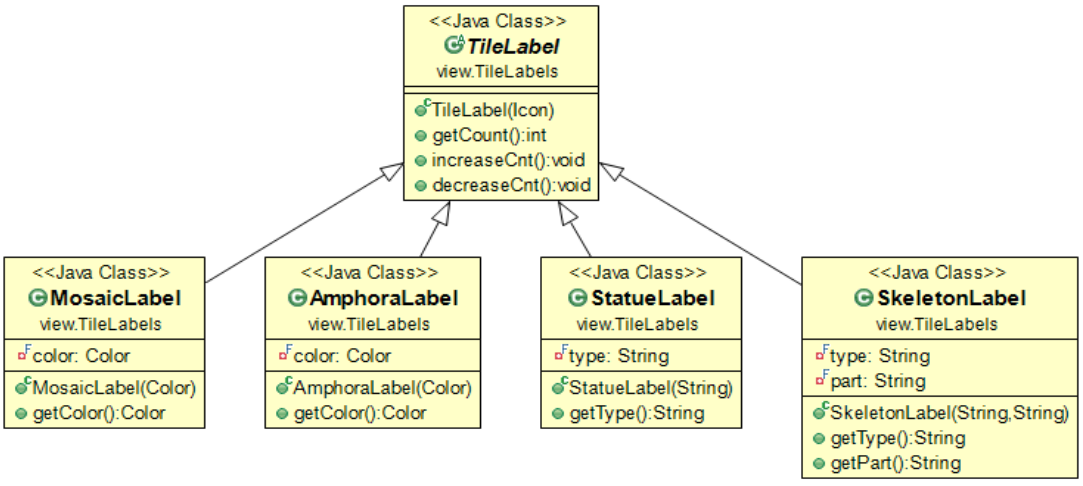

<<Java Class>>
AmphipolisFrame
view
AmphipolisFrame()
getBoard():BoardPane
getInfoPanel():PlayerInfoPanel
getPlayerTiles():PlayerTilesGrid
-board
0..1
-infoPanel
0..1
-playerTiles
0..1
<<Java Class>>
BoardPane
view
BoardPane()
setBoardBg():void
addTileAreasToBoard():void
getMosaicAreaGrid():TileArea
getAmphoraAreaGrid():TileArea
getStatueAreaGrid():TileArea
getSkeletonAreaGrid():TileArea
getLandslideAreaGrid():TileArea
<<Java Class>>
PlayerInfoPanel
view
playerName: JLabel
useCardText: JLabel
drawBtn: JButton
endTurnBtn: JButton
assistantCard: JLabel
diggerCard: JLabel
archaeologistCard: JLabel
professorCard: JLabel
PlayerInfoPanel()
disableCard(String):void
enableCard(String):void
setPlayerName(String):void
getDrawBtn():JButton
getEndTurnBtn():JButton
getAssistantCard():JLabel
getDiggerCard():JLabel
getArchaeologistCard():JLabel
getProfessorCard():JLabel
<<Java Class>>
PlayerTilesGrid
view
PlayerTilesGrid()
initializePlayerTileLabels():void
getPlayerTiles():TileLabel[]
0..1
<<Java Class>>
TileArea
view
category: String
TileArea(int,String)
addTileLabelsToGrid():void
populateTileLabels():void
getTileLabels():TileLabel[]
getCategory():String
-playerTiles
0..*
-tileLabels
0..*
<<Java Class>>
TileLabel
view.TileLabels
TileLabel(Icon)
getCount():int
increaseCnt():void
decreaseCnt():void
<<Java Class>>
MosaicLabel
view.TileLabels
color: Color
MosaicLabel(Color)
getColor():Color
<<Java Class>>
AmphoraLabel
view.TileLabels
color: Color
AmphoraLabel(Color)
getColor():Color
<<Java Class>>
StatueLabel
view.TileLabels
type: String
StatueLabel(String)
getType():String
<<Java Class>>
SkeletonLabel
view.TileLabels
type: String
part: String
SkeletonLabel(String,String)
getType():String
getPart():String

4.1 Η κλάση TileLabel και οι υποκλάσεις της

Όπως αναφέρθηκε προηγουμένως, η κλάση αυτή αναπαριστά ένα πλακίδιο του παιχνιδιού και μπορεί να οριστεί ως υποκλάση της Jlabel. Τα βασικά της στοιχεία είναι μια εικόνα που αντιστοιχεί στο πλακίδιο, και μια ετικέτα που θα λειτουργεί σαν μετρητής που μετράει πόσα πλακίδια του εκάστοτε τύπου υπάρχουν (στο ταμπλό ή στο χέρι του παίχτη).

Είδαμε όμως νωρίτερα, ότι στο πακέτο Model, υπάρχουν διάφορα είδη πλακιδίων και για τον λόγο αυτό (και έναν ακόμα που θα αναφέρουμε σε λίγο), η κλάση αυτή ορίζεται ως αφηρημένη. Με αυτόν τον τρόπο, τα κοινά γνωρίσματα και λειτουργίες ενός πλακιδίου συγκεντρώνονται σε ένα μέρος και ορίζονται μια φορά, ενώ η κάθε υποκλάση μπορεί να ορίσει τις διαφορές της. Ο άλλος λόγος που χρειάζονται υποκλάσεις για τις διάφορες κατηγορίες πλακιδίων, είναι διότι κάποια πλακίδια έχουν χρώματα (αμφορείς, μωσαικά), κάποια έχουν μέγεθος (σκελετοί), ενώ κάποια άλλα έχουν απλά τύπο (αγάλματα).

Χωρίζοντας τα σε υποκλάσεις λοιπόν, μπορούμε να ορίσουμε πως θα διαχειριζόμαστε αυτές τις πληροφορίες για την δημιουργία των πλακιδίων. Συγκεκριμένα, θα δούμε παρακάτω πως αυτά τα στοιχεία, χρησιμοποιούνται κυρίως για την εύρεση της εικόνας που αντιστοιχεί στο εκάστοτε πλακίδιο αλλά και στην αναγνώριση του ποιο πλακίδιο της διεπαφής επέλεξε να τραβήξει ο παίκτης από το ταμπλό μιας και ο μόνος τρόπος να ξεχωρίσουμε τα πλακίδια μωσαϊκών, είναι μέσω του χρώματός τους. Ακολουθεί το UML διάγραμμα της TileLabel και των υποκλάσεων της καθώς και οι μέθοδοι της TileLabel (δεν έχει γνωρίσματα).



Methods:

public TileLabel(Icon icon); // “Constructor” used to initialize the label/counter and its image using the passed icon

public int getCount(); // Accessor : returns the text of the tile label as a number

public void increaseCnt(); // Transformer: updates the label’s count by increasing it by 1

public void decreaseCnt(); // Transformer: updates the label’s count by decreasing it by 1

4.1.1 Η υποκλάση MosaicLabel

Αυτή η κλάση αναπαριστά ένα πλακίδιο μωσαϊκού και για να μπορούμε να ξεχωρίζουμε ποιο μωσαϊκό πλακίδιο τραβάει ο παίχτης από το ταμπλό όταν πατάει πάνω του, δέχεται σαν όρισμα μέσω της κατασκευάστριας μεθόδου του ένα χρώμα το οποίο χρησιμοποιείται επίσης και για την εύρεση της κατάλληλης εικόνας. Ακολουθούν τα γνωρίσματα και οι μέθοδοι της κλάσης.

Attributes:

private final Color color; // The mosaic tile’s color

Methods:

public MosaicLabel(Color color); // Constructor: sets the image of the Jlabel based on the given color through the super class & initializes the color field

public Color getColor(); // Accessor: returns the mosaic label’s color

4.1.2 Η υποκλάση AmphoraLabel

Η κλάση αυτή αναπαριστά ένα πλακίδιο αμφορέα και για να μπορούμε να ξεχωρίζουμε και εδώ ποιο πλακίδιο αμφορέα τραβάει ο παίχτης από το ταμπλό όταν πατάει πάνω του, δέχεται σαν όρισμα μέσω της κατασκευάστριας μεθόδου του ένα χρώμα το οποίο χρησιμοποιείται επίσης και για την εύρεση της κατάλληλης εικόνας. Ακολουθούν τα γνωρίσματα και οι μέθοδοι της κλάσης.

Attributes:

private final Color color; // The amphora tile’s color

Methods:

public AmphoraLabel(Color color); // Constructor: sets the image of the Jlabel based on the given color through the super class & initializes the color field

public Color getColor(); // Accessor: returns the amphora label's color

4.1.3 Η υποκλάση StatueLabel

Η κλάση αυτή αναπαριστά ένα πλακίδιο αγάλματος και για να μπορούμε να ξεχωρίζουμε και εδώ ποιο πλακίδιο αγάλματος τραβάει ο παίχτης από το ταμπλό όταν πατάει πάνω του, δέχεται σαν όρισμα μέσω της κατασκευάστριας μεθόδου του έναν τύπο (π.χ. Sphynx ή Caryatid) , ο οποίος χρησιμοποιείται επίσης και για την εύρεση της κατάλληλης εικόνας. Ακολουθούν τα γνωρίσματα και οι μέθοδοι της κλάσης.

Attributes:

private final String type; // The type of the statue label

Methods:

public StatueLabel(String type); // Constructor: sets the image of the Jlabel based on the given type through the super class & initializes the type field

public String getType(); // Accessor: returns the statue label's type

4.1.4 Η υποκλάση SkeletonLabel

Η κλάση αυτή αναπαριστά ένα πλακίδιο σκελετού και για να μπορούμε να ξεχωρίζουμε ποιο πλακίδιο σκελετού τραβάει ο παίχτης από το ταμπλό όταν πατάει πάνω του, δέχεται σαν ορίσματα μέσω της κατασκευάστριας μεθόδου του έναν τύπο (π.χ. Big ή Small) και το τμήμα του σκελετού (Top ή Bottom) , τα οποία χρησιμοποιεί επίσης και για την εύρεση της κατάλληλης εικόνας. Ακολουθούν τα γνωρίσματα και οι μέθοδοι της κλάσης.

Attributes:

private final String type; // The skeleton's type

private final String part; // The skeleton's part

Methods:

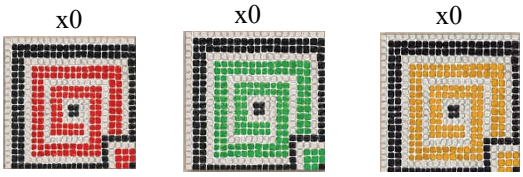
public SkeletonLabel(String type, String part); // Constructor : sets the image of the Jlabel based on the given parameters through the super class & initializes the fields

public String getType(); // Accessor: returns the skeleton label's type

public String getPart(); // Accessor: returns the skeleton label's part

4.2 Η κλάση TileArea

Η TileArea αναπαριστά μια περιοχή τοποθέτησης πλακιδίων στο ταμπλό και είναι τύπου Jpanel. Για να τοποθετούνται κατάλληλα τα πλακίδια τύπου TileLabel στο Jpanel, χρησιμοποιείται ως LayoutManager ένα GridLayout, το οποίο θα έχει μόνο μια γραμμή και αριθμό στηλών αντίστοιχο του πλήθους των διαφορετικών πλακιδίων του κάθε τύπου. Για παράδειγμα, ένα μωσαϊκό βάση του χρώματος του χωρίζεται σε τρία χρώματα. Επομένως, το Grid για μια τέτοια περιοχή θα είναι όπως το παρακάτω:



Ακολουθούν τα γνωρίσματα και οι μέθοδοι της κλάσης:

Attributes:

private final TileLabel[] tileLabels; // the list of TileLabels that the area holds

private final String category; // The category of the area specifying what type of TileLabels the area will hold, i.e. Mosaic, Amphora etc.

Methods:

public TileArea(int numOfTilesToDisplay, String category); // Constructor which creates and sets the layout manager for the panel by using the numOfTilesToDisplay as the column number for the grid & performs needed initializations

private void addTileLabelsToGrid(); // adds each tile label from the array to the panel's grid

private void populateTileLabels(); // creates the necessary number of Tile labels based on the category and adds them to the array

public TileLabel[] getTileLabels(); // Accessor: returns the tile labels of the area

public String getCategory(); // Accessor: returns the category of the area

4.3 Η κλάση PlayerTilesGrid

Αυτή η κλάση αναπαριστά την συλλογή πλακιδίων ενός παίχτη. Είναι τύπου Jpanel και χρησιμοποιεί ως LayoutManager ένα GridLayout για τη διάταξη των πλακιδίων. Το Grid θα έχει μία μόνο γραμμή και μία στήλη για κάθε πλακίδιο του παιχνιδιού που μπορεί να κρατήσει ένας παίχτης (14 διαφορετικά πλακίδια). Τα γνωρίσματα και οι μέθοδοι της κλάσης δίνονται παρακάτω:

Attributes:

private final TileLabel[] playerTiles; // The player's tile collection

Methods:

public PlayerTilesGrid(); // Constructor which creates and sets the layout manager for the jpanel, other attributes such as size, and performs any needed initialization

private void initializePlayerTileLabels(); // Creates a tile label for each type of tile for the player's collection, adds it to the playerTiles array and then adds it to the grid.

public TileLabel[] getPlayerTiles(); // Accessor: returns the player's tile collection in the form of tile labels

4.4 Η κλάση PlayerInfoPanel

Η PlayerInfoPanel αναπαριστά την περιοχή της γραφικής διεπαφής που περιέχει επιπλέον πληροφορίες για τον εκάστοτε παίχτη καθώς και κάποια διαδραστικά κομμάτια. Πιο συγκεκριμένα, περιέχει ενα Jlabel για το όνομα του παίχτη, ένα για τον τίτλο της περιοχής καρτών, 4 Jlabels για τις διάφορες κάρτες του κάθε παίχτη (τις οποίες μπορεί να παίξει πατώντας πάνω τους) και δύο Jbuttons, ένα για να τραβήξει πλακίδια από την σακούλα και ένα για να τερματίσει την σειρά του. Για να τοποθετηθούν όλα αυτά στο τελικό Jframe , η κλάση ορίζεται ως υποκλάση της Jpanel και για LayoutManager χρησιμοποιούμε ένα GridBagLayout καθώς μας

επιτρέπει να διατάξουμε τα στοιχεία του όπως σε ενα GridLayout με το επιπλέον προνόμιο ότι κάποια στοιχεία (π.χ. τα Jbuttons) , μπορούν να παίρνουν περισσότερο χώρο από κάποια άλλα. Τα γνωρίσματα και οι μέθοδοι της κλάσης δίνονται παρακάτω:

Attributes:

private final Jlabel playerName; // holds the active player's name

private final Jlabel useCardText; // acts as the title for the card collection area

private final Jbutton drawBtn; // The "Draw" button that allows players to draw tiles from the bag

private final Jbutton endTurnBtn; // The "End Turn" button that allows players to end their turn

private final Jlabel assistantCard; // Represents the player's assistant card

private final Jlabel diggerCard; // Represents the player's digger card

private final Jlabel professorCard; // Represents the player's professor card

private final Jlabel archaeologistCard; // Represents the player's archaeologist card

Methods:

public PlayerInfoPanel(); // Constructor : creates and sets the layout manager for the panel, performs actions such as setting the size of the panel and initializes & places all the components in the layout.

public void disableCard(String cardName); // Finds the card matching the given name and prevents the player from playing the card again while also visually disabling it.

public void enableCard(String cardName); // Finds the card matching the given name and visually enables it thus allowing the player to play it

public void setPlayerName(String name); // Update's the respective jlabel's text with the given player's name

public Jbutton getDrawBtn(); // Accessor: returns the Jbutton representing the Draw button of the game

public JButton getEndTurnBtn(); // Accessor: returns the JButton representing the End turn button of the game

public JLabel getAssistantCard(); // Accessor: returns the JLabel representing the player's assistant card

public JLabel getDiggerCard(); // Accessor: returns the JLabel representing the player's digger card

public JLabel getProfessorCard(); // Accessor: returns the JLabel representing the player's professor card

public JLabel getArchaeologistCard(); // Accessor: returns the JLabel representing the player's archaeologist card

4.5 Η κλάση BoardPane

Με αυτή την κλάση αναπαριστούμε το ταμπλό του παιχνιδιού. Το ταμπλό όπως έχουμε ήδη αναφέρει περιέχει 5 περιοχές τοποθέτησης πλακιδίων και για τον λόγο αυτό αποτελεί υποκλάση της JLayeredPane καθώς το ταμπλό θα έχει ένα επίπεδο για το background του και ένα για τις διάφορες περιοχές που τοποθετούνται επί αυτού. Ακολουθούν τα γνωρίσματα και οι μέθοδοι της κλάσης:

Attributes:

private final TileArea mosaicArea; // the tile area of the board that holds mosaic tiles

private final TileArea amphoraArea; // the tile area of the board that holds amphora tiles

private final TileArea statueArea; // the tile area of the board that holds statue tiles

private final TileArea skeletonArea; // the tile area of the board that holds skeleton tiles

private final TileArea landslideArea; // the tile area of the board that holds landslide tiles

Methods:

public BoardPane(); // Constructor : performs necessary setup (size, position of elements etc.)

private void setBoardBg(); // sets the board's background

public void addTileAreasToBoard(); // adds the various tile areas on the board and positions them

public TileArea getMosaicArea(); // Accessor: returns the mosaic tile area

public TileArea getAmphoraArea(); // Accessor: returns the amphora tile area

public TileArea getStatueArea(); // Accessor: returns the statue tile area

public TileArea getSkeletonArea(); // Accessor: returns the skeleton tile area

public TileArea getLandslideArea(); // Accessor: returns the landslide tile area

4.6 Η κλάση AmphipolisFrame

Αυτή αποτελεί την τελευταία κλάση του πακέτου View και είναι υπεύθυνη για την συγκέντρωση, διάταξη και εμφάνιση όλων των προηγούμενων κλάσεων του πακέτου που είδαμε, με σκοπό την επίτευξη μιας λειτουργικής γραφικής διεπαφής για το παιχνίδι. Όντας η κλάση που θα εμφανίσει όλη την διεπαφή, ορίζεται ως υποκλάση της JFrame με LayoutManager ένα BorderLayout έτσι ώστε το ταμπλό να τοποθετηθεί στο κέντρο (BorderLayout.CENTER), το PlayerInfoPanel στα δεξιά (BorderLayout.EAST) και το PlayerTilesGrid κάτω (BorderLayout.SOUTH) από αυτά τα δύο. Τέλος, η κλάση εξάγει μέσω των μεθόδων της τα επιμέρους components της που προαναφέραμε, με σκοπό η κλάση Controller να μπορεί να τα προσπελάσει και να τα τροποποιήσει κατάλληλα. Ακολουθούν τα γνωρίσματα και οι μέθοδοι της.

Attributes:

private final BoardPane board; // the UI class that represents the board of the game

private final PlayerInfoPanel infoPanel; // the UI class that represents the player info panel

private final PlayerTilesGrid playerTiles; // the UI class that represents the player's tile collection

Methods:

public AmphipolisFrame(); // Constructor : creates and sets the layout manager for the frame and performs all necessary initializations

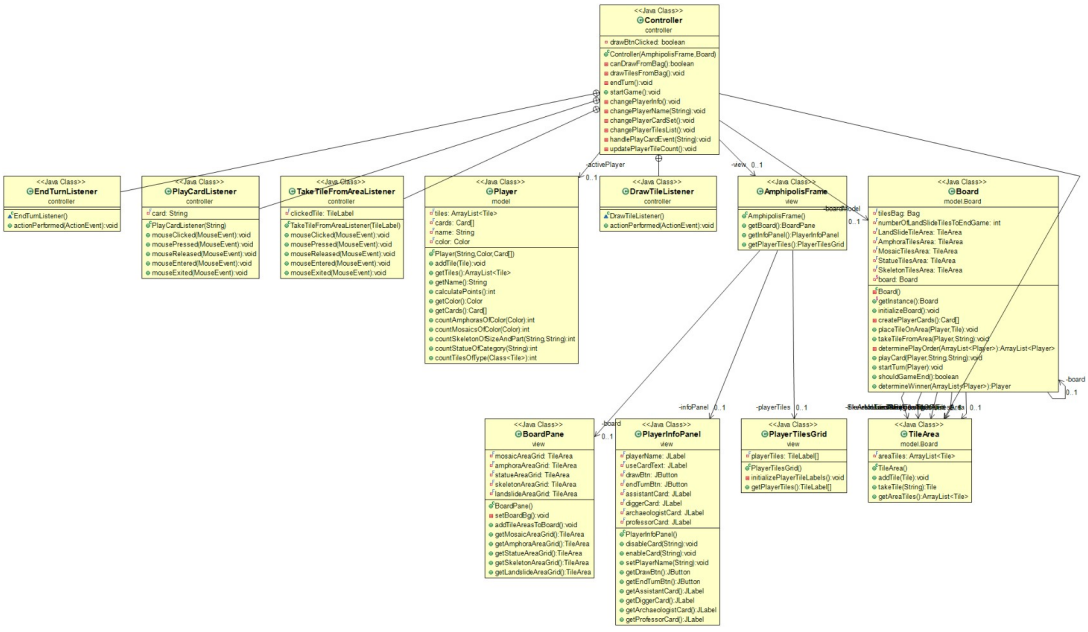
public BoardPane getBoard(); // Accessor: returns the UI component representing the board of the game

public PlayerInfoPanel getInfoPanel(); // Accessor: returns the UI component representing the player's info panel

public PlayerTilesGrid getPlayerTiles(); // Accessor: returns the UI component representing the player's tile collection

5. Η Αλληλεπίδραση μεταξύ των κλάσεων – Διαγράμματα UML

Είδαμε στις προηγούμενες ενότητες πως επικοινωνούν μεταξύ τους οι διάφορες κλάσεις του εκάστοτε πακέτου και γιατί. Σε αυτήν την ενότητα θα δούμε μέσω διαγραμμάτων UML, πως επικοινωνούν μεταξύ τους οι κλάσεις διαφορετικών πακέτων, με τέτοιο τρόπο ώστε να επιτευχθεί η σωστή υλοποίηση της εργασίας διατηρώντας την αρχιτεκτονική MVC.



Στο παραπάνω UML diagram , βλέπουμε πως επιτυγχάνεται η αρχιτεκτονική MVC όπου η κλάση Controller είναι ο εγκέφαλος του παιχνιδιού και είναι αυτή η οποία ελέγχει και το View και το Model. Όταν είδαμε τα UML diagrams των πακέτων View & Model , είδαμε πως κανένα εκ των δυο δεν γνωρίζει για το άλλο. Μόνο ο Controller γνωρίζει για την ύπαρξη και των δυο και μέσω της διεπαφής που

δημιουργούν με τις μεθόδους τους, μπορεί να τα ελέγξει και να τα τροποποιήσει όπως κρίνεται καταλληλότερο.

Συγκεκριμένα, η Controller επικοινωνεί με τις κλάσεις Board, Player & TileArea του μοντέλου και με την κλάση AmphipolisFrame της γραφικής διεπαφής. Η κάθε μία κλάση του κάθε πακέτου, εξάγει όπως είδαμε τρόπους και αντικείμενα, τα οποία μας επιτρέπουν να ανανεώνουμε τα δεδομένα του model όταν ο χρήστης ξεκινάει κάποιο γεγονός μέσω του View και αντιστρόφως. Τέλος, οι εσωτερικές κλάσεις της Controller που βλέπουμε , είναι ένα πολύ σημαντικό κομμάτι του MVC, διότι με τον τρόπο αυτό η κλάση μπορεί να ορίσει η ίδια σε ποια γεγονότα θα αντιδρά το View , καθώς και το πότε και το τι θα γίνεται όταν συμβαίνει αυτό. Έτσι , το view και το model δεν επικοινωνούν ποτέ απευθείας μεταξύ τους, παρά μόνο μέσω του controller, ο οποίος γνωρίζει για την ύπαρξη και τις ικανότητες και των δύο, επιτυγχάνοντας έτσι, την επιθυμητή αρχιτεκτονική της εργασίας.

6. Λειτουργικότητα (Β Φάση)

Σε αυτήν την ενότητα θα αναλυθεί η τελική υλοποίηση της εργασίας, συμπεριλαμβανομένου των προβλημάτων που αντιμετωπίστηκαν , των σχεδιαστικών και προγραμματιστικών αποφάσεων που λήφθησαν και πως αυτές επηρεάζουν την εμπειρία του χρήστη, τυχόν διαφοροποιήσεις με την αρχική σχεδίαση της Α φάσης και ότι άλλο κρίνεται απαραίτητο.

6.1 Σχεδιαστικές διαφοροποιήσεις με τη σχεδίαση της Α φάσης

6.1.1 Πακέτο Model

A) Κλάση Player

Μερικές διαφοροποιήσεις σε αυτήν την κλάση είναι ότι προστέθηκε το γνώρισμα int finalPoints καθώς και οι αντίστοιχες accessor & transformer μέθοδοι. Το γνώρισμα αυτό χρησιμοποιείται από την κλάση Board του μοντέλου για να υπολογίσει τους πόντους του κάθε παίχτη όταν τελειώσει το παιχνίδι και να ενημερώσει το πεδίο αυτό.

Επίσης, προστέθηκαν οι μέθοδοι getSkeletonPoints(), getAmphoraPoints() & getMosaicPoints() οι οποίες χρησιμοποιούνται από την calculatePoints() και η λειτουργία τους είναι αυτονόητη.

Τέλος, αφαιρέθηκε η μέθοδος countTilesOfType(Class) διότι τελικά δεν ήταν χρήσιμη για την τελική υλοποίηση της εργασίας.

B) Κλάση TileArea και οι νέες υποκλάσεις της

Ίσως η σημαντικότερη αλλαγή στο πακέτο αυτό είναι η παραμετροποίηση της κλάσης TileArea ως TileArea<T extends Tile> και η δήλωση της ως αφηρημένη. Με τον τρόπο αυτό, κοινές μέθοδοι και γνωρίσματα μπορούν να κληρονομηθούν από τις νέες τώρα υποκλάσεις (AmphoraTileArea, LandslideTileArea, MosaicTileArea, SkeletonTileArea και StatueTileArea) και λειτουργίες που απαιτούν διαφορετικό χειρισμό βάση του τύπου του πλακιδίου, όπως η αφαίρεση πλακιδίων από την περιοχή, μπορούν να τις χειριστούν οι υποκλάσεις ξεχωριστά.

Η κλάση TileArea λοιπόν, περιέχει πλεόν τα εξής:

Attributes:

private final ArrayList<T> areaTiles; // holds the tiles of the area of type <T>

Methods:

public TileArea(); // empty default constructor

public void addTile(T tile); // adds a tile of type T to the area

public ArrayList<T> getAreaTiles(); // accessor function which returns the areas <T> type tiles

Οι υποκλάσεις της που αναφέρθηκαν παραπάνω, περιέχουν η καθεμία από δύο μεθόδους (πέρα του default constructor) η οποίες εκτελούν τη λειτουργία της αφαίρεσης πλακιδίου και της καταμέτρησης πλακιδίων βάση χαρακτηριστικών του αντίστοιχου τύπου.

Δίνεται ένα παράδειγμα μέσω της κλάσης AmphoraTileArea η οποία είναι υποκλάση της TileArea<Amphora>. Οι υπόλοιπες είναι παρόμοιες και μπορούν να αναλυθούν εις βάθος μέσω του JavaDoc φακέλου που παραδόθηκε με την εργασία.

Methods:

public Amphora takeAmphoraTile(String color); // retrieves and returns an amphora tile matching the given color

public int getAmphoraCountOfColor(String color); // counts and returns the number of amphora tiles with the given color

C) Η κλάση Bag

Σε αυτήν προστέθηκε απλώς η μέθοδος public Tile drawTileOfType(Class<? Extends Tile>) η οποία χρησιμοποιείται από την κλάση Board που θα δούμε παρακάτω για το τράβηγμα ενός πλακιδίου του κάθε τύπου κατά την εκκίνηση του παιχνιδιού.

D) Η κλάση Board

Εδώ αρχικά τα πεδία τύπου TileArea που αντιπροσώπευαν τις διάφορες περιοχές τοποθέτησης πλακιδίων, έχουν μετατραπεί στις αντίστοιχες υποκλάσεις της TileArea όπως φαίνεται παρακάτω:

private final LandslideTileArea;

private final AmphoraTileArea;

private final MosaicTileArea;

private final SkeletonTileArea;

private final StatueTileArea;

Ακολουθούν και οι μέθοδοι της κλάσης:

public static Board getInstance(); // returns the instance of the class

private Board(); // private default constructor

public void initializeBoard(ArrayList<Player>); // performs initialization actions, see javadoc

private Card[] createPlayerCards(Player); // creates a card set for the given player/owner and returns it

public void takeTileFromArea(Player player, Tile tile); // removes a tile of the given type from the appropriate area

private void determinePlayOrder(ArrayList<Player>); // determines the player order

public void draw(); // draws 4 tiles from the bag and places them in the appropriate area

public void drawAndPlaceInitialTiles(); // draws one tile of each type from the bag and places them in the appropriate areas

private void addTileToArea(Tile); // adds the given tile to the appropriate area

public boolean shouldGameEnd(); // returns true if the game should end or false otherwise

public void playCard(Player,Card); // plays the given card from the given player's hand (if not played already)

public void determineWinner(ArrayList<Player>); // counts and updates each player's points in order to determine the winner

public LandslideTileArea(); // accessor function for the landslide tile area field

public AmphoraTileArea(); // accessor function for the amphora tile area field

public MosaicTileArea(); // accessor function for the mosaic tile area field

public StatueTileArea(); // accessor function for the statue tile area field

public SkeletonTileArea(); // accessor function for the skeleton tile area field

Ακολουθούν οι αλλαγές που έχουν γίνει σε σχέση με τη σχεδίαση της κλάσης κατά την Α φάση:

1) Η μέθοδος placeTileOnArea(Player,Tile) της φάσης Α έχει μετατραπεί στην addTileToArea(Tile)

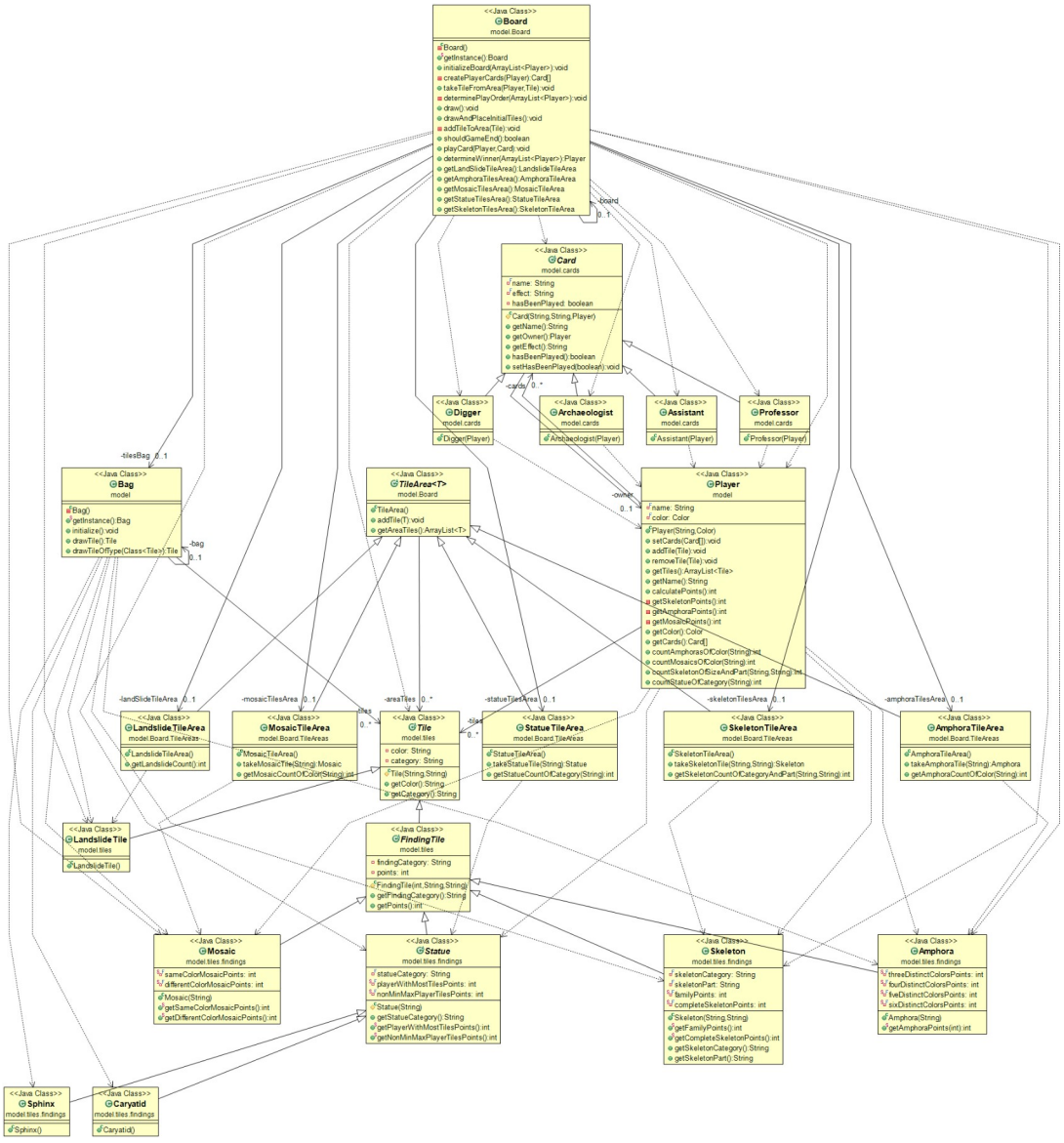
2) Η takeTileFromArea(Player,String) της Α φάσης μετατράπηκε στην takeTileFromArea(Tile)

3) Η playCard(Player,String,String) της φάσης Α άλλαξε στην playCard(Player,Card)

4) Η μέθοδος startTurn(Player) της Α φάσης αφαιρέθηκε

5) Προστέθηκαν οι μέθοδοι draw(), drawAndPlaceInitialTiles()

Παρατίθεται επίσης το νέο UML διάγραμμα του πακέτου;



6.1.2 Πακέτο View

A) Κλάση TileArea και υποκλάσεις

Εδώ μια ιδιαίτερα σημαντική αλλαγή είναι η μετατροπή της κλάσης TileArea σε αφηρημένη κλάση με σκοπό την επαναχρησιμοποίηση χαρακτηριστικών και μεθόδων από τις υποκλάσεις της (AmporaTileAreaView,LandslideTileAreaView,MosaicTileAreaView,SkeletonTileAreaView & StatueTileAreaView). Πιο συγκεκριμένα, η κλάση αυτή μέσω του constructor της δημιουργεί και ορίζει σαν layout manager της (ας θυμηθούμε οτι αποτελεί υποκλάση Jpanel) ενα GridLayout και καθορίζει και άλλα χαρακτηριστικά που σχετίζονται με την εμφλανιση του.

Τέλος, το χαρακτηριστικό String category και οι σχετικές μεθόδους του έχουν αφαιρεθεί μιας και πλέον τα ειδικά χαρακτηριστικά της κάθε περιοχής καθορίζονται στις υποκλάσεις της και έχει επίσης προστεθεί η void setTileLabels(TileLabel[]) μέθοδος .

Όσον αφορά τις υποκλάσεις της TileArea, όλες τους μέσω του constructor καθορίζουν το μέγεθος του GridLayout της υπερκλάσης, δηλώνουν και γεμίζουν έναν πίνακα από TileLabels του εκάστοτε τύπου μέσω της void populateTileLabels() και τις προσθέτουν στο grid.

B) Υποκλάσεις της TileLabel

Η κάθε υποκλάση της TileLabel στην Β φάση της εργασίας πλέον κάνει Override την μέθοδο equals(Object) της Object και συγκρίνει αν το αντικείμενο που της περάσαμε είναι “ίδιο” με το αντικείμενο μέσω του οποίου την καλέσαμε συγκρίνοντας απλά αν είναι ίδια τα χαρακτηριστικά του εκάστοτε τύπου. Για παράδειγμα, η StatueLabel υποκλάση, ελέγχει αν τα δυο αντικείμενα έχουν τον ίδιο τύπο (Sphinx/Caryatid) και αναλόγως επιστρέφει την κατάλληλη τιμή.

Αυτή η αλλαγή έγινε για να απλοποιήσει κάποιους ελέγχους που θα δούμε αργότερα στην κλάση Controller.

C) Η κλάση BoardPanel

Σε σχέση με την σχεδίαση της στην Α φάση, η κλάση πλέον αντί για τα 5 TileArea γνωρίσματα της, στην φάση Β έχει τα ίδια γνωρίσματα αλλά με τον αντίστοιχο τύπο της κάθε περιοχής. Δηλαδή έχει τα παρακάτω:

```
private final MosaicTileAreaView mosaicArea;

private final AmphoraTileAreaView amphoraArea;

private final StatueTileAreaView statueArea;

private final SkeletonTileAreaView skeletonArea;

private final LandslideTileAreaView landslideArea;
```

Τα υπόλοιπα χαρακτηριστικά της παραμένουν ίδια.

D) Η κλάση PlayerInfoPanel

Σε αυτή την κλάση, κατά την Β φάση προστέθηκε ένα ακόμα Jbutton γνώρισμα το stopCardEffectBtn του οποίου η λειτουργία θα αναλυθεί παρακάτω και έχει να κάνει με προγραμματιστικές επιλογές που πάρθηκαν κατά την φάση της υλοποίησης.

Επίσης, προστέθηκαν οι μέθοδοι :

private void setCardImages(); // sets each card's image and size

private void populateLayout(); // adds all the components to the class's layout manager and styles them

private void setUpCard(String cardName, boolean hasBeenPlayed); // visually disables or enables a card label based on the value of the hasBeenPlayed param.

Private JButton getStopCardEffectBtn(); // accessor function which returns the stopCardEffectBtn instance

Εκτός των παραπάνω, δεν έγινε κάποια επιπλέον αλλαγή.

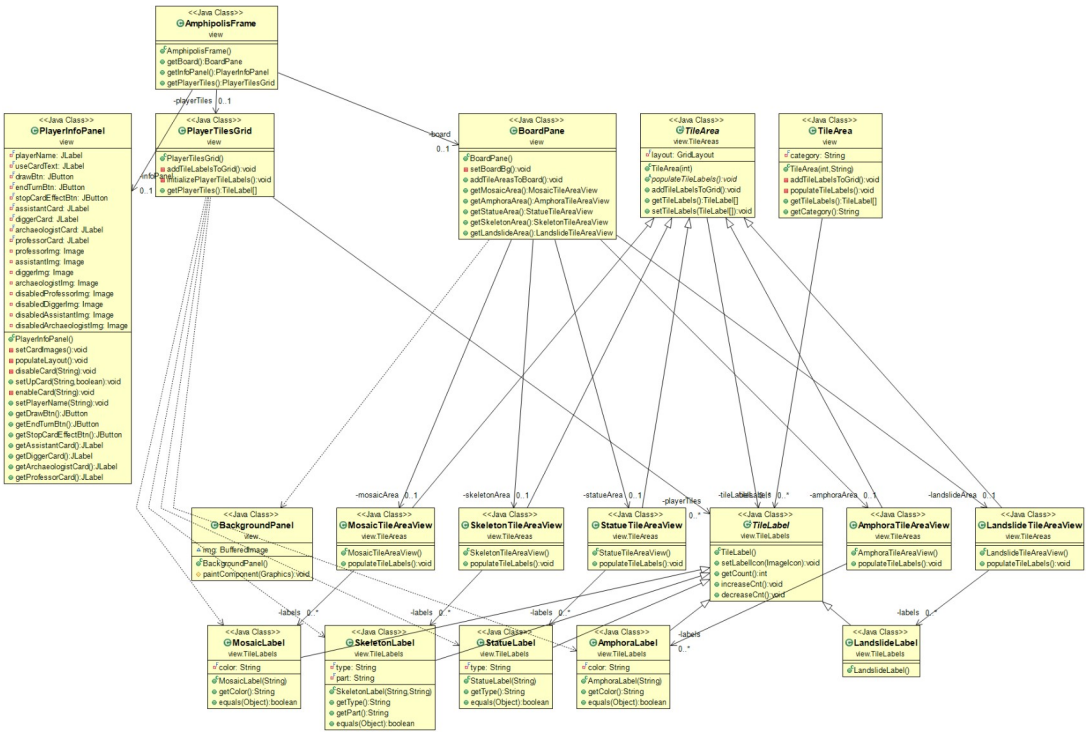
E) Η κλάση PlayerTilesGrid

Εδώ, η μόνη αλλαγή σε σχέση με την σχεδίαση της Α φάσης είναι η προσθήκη της μεθόδου private void addTileLabelsToGrid() η οποία απλώς προσθέτει όλα τα TileLabel components στο Grid Layout της κλάσης.

F) Η νέα κλάση BackgroundPanel

Τέλος, στο πακέτο έχει επιπλέον προστεθεί η κλάση BackgroundPanel η οποία αποτελεί υποκλάση της JPanel και σκοπός της είναι να χρησιμοποιηθεί ως το background του ταμπλό. Το μόνο γνώρισμα της είναι το BufferedImage img; το οποίο αρχικοποιείται μέσω της κατασκευάστριας μεθόδου της με την κατάλληλη εικόνα. Τέλος, μέσω της Overriden μεθόδου protected void paintComponent(Graphics g), η εικόνα ζωγραφίζεται σαν το background του Jpanel με τα κατάλληλα χαρακτηριστικά.

Ακολουθεί το νέο UML διάγραμμα του πακέτου :



6.1.3 Πακέτο Controller

Στην κλάση Controller του πακέτου έγιναν αρκετές προσθέσεις νέων γνωρισμάτων καθώς κατά την σχεδίαση της στην Α φάση δεν είχα υπολογίσει πως θα χειριζόμουν το γεγονός όπου κάποιος παίζει μια κάρτα και πως η ιδιότητα της κάθε κάρτας θα εφαρμοζόταν. Προστέθηκαν επίσης και κάποιες επιπλέον μέθοδοι για την ενημέρωση της γραφικής διεπαφής και μια ακόμη εσωτερική κλάση, η StopCardEffectListener που υλοποιεί την ActionListener διεπαφή και χρησιμοποιείται ως action listener του stopCardEffectBtn Jbutton της κλάσης PlayerInfoPanel του πακέτου View. Ακολουθούν τα νέα γνωρίσματα και οι μέθοδοι που προστέθηκαν κατά την Β φάση καθώς και σύντομες περιγραφές του καθενός.

Attributes:

private Card activeCard; // the currently active card (if any)

private boolean isCardEffectActive; // when a card has been played and the effect hasn't been executed fully, this will be true

private boolean cardUsedThisRound; // true when a card has been played this round

private int remainingTilesThisRound; // keeps track of how many tiles the player can draw each turn from an area (based on the rules of the game)

private int extraTilesThisRound; // keeps track of how many extra tiles the player can draw this turn (Based on the effect of a card)

// the following attributes keep track of how many extra tiles of each type the player is //allowed to take this round when playing a card that allows taking tiles from multiple areas

private int cardEffectExtraAmphoras;

private int cardEffectExtraMosaics;

private int cardEffectExtraSkeletons;

private int cardEffectExtraStatues;

private int extraTilesFromSameArea; // used by the assistant and archaeologist card effects to determine how many extra tiles the player can take from the same area

Methods:

private void displayWinner(); // responsible for displaying the winner and all players along with their points in a dialog

private void updateTileCounts(); // updates the view's tile area tile counts

private void enableCardEffect(String cardName); // performs all necessary actions to enable the card's effect that matches the given cardName

private void disableCardEffect(); // performs all necessary actions to stop the active card's effect

public boolean getCanDraw(); // accessor function which returns the value of the canDraw field

public void setCanDraw(boolean canDraw); // transformer function which sets the value of the canDraw field to the given one

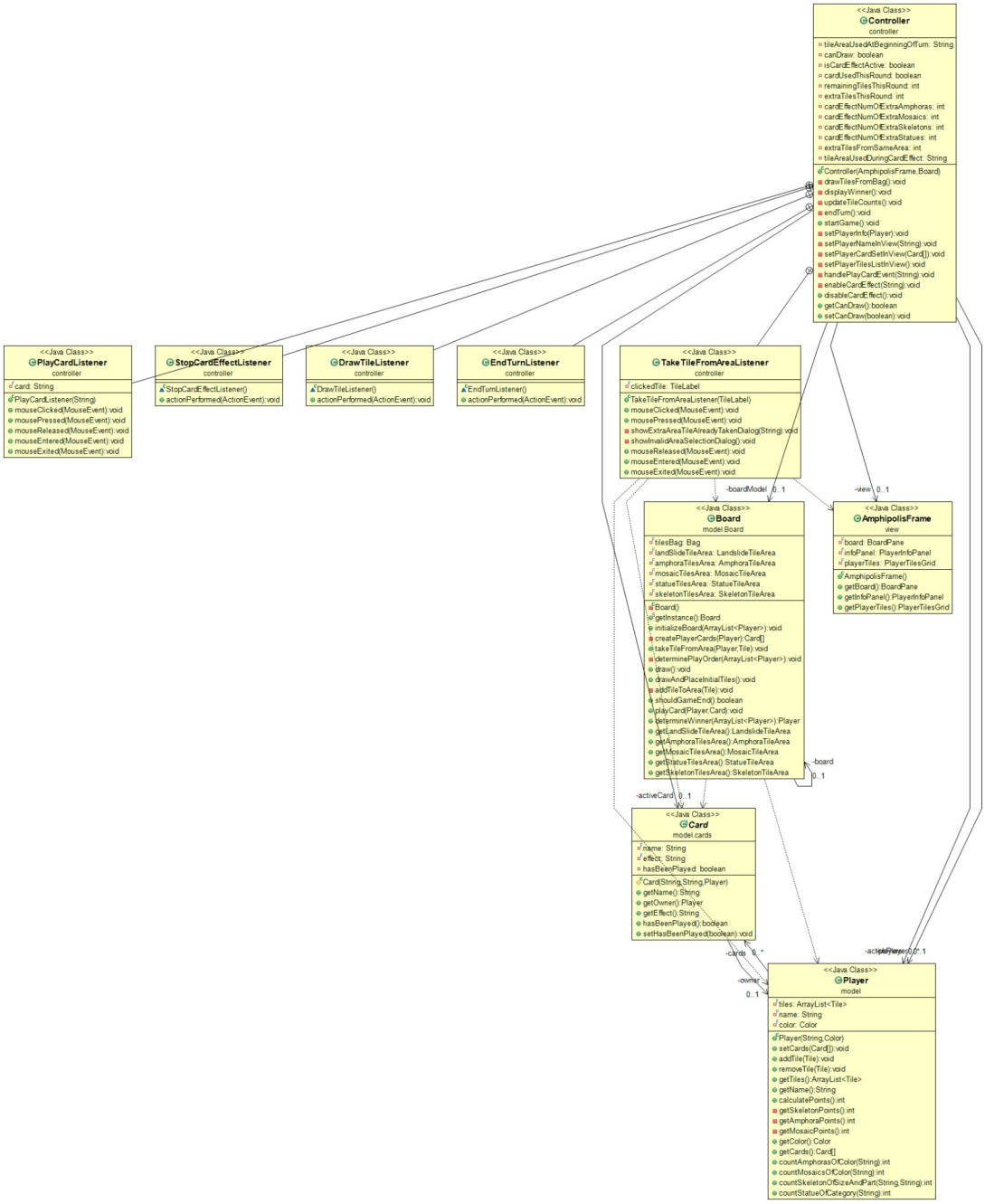
Τέλος, έχουν γίνει οι παρακάτω αλλαγές ακόμη:

1) Το πεδίο drawBtnClicked μετατράπηκε στο canDraw

2) Το πεδίο tileAreaUsedAtBeginningOfTurn μετατράπηκε από TileArea τύπο σε String

3) και οι μέθοδοι changePlayerInfo(),changePlayerName(),changePlayerCardSet() & changePlayerTileList() άλλαξαν σε setPlayerInfo(Player), setPlayerNameInView(String), setPlayerCardSetInView(Card[]) & setPlayerTilesListInView.

Ακολουθεί το UML διάγραμμα του πακέτου:



6.1.4 Πακέτο Exceptions

Κατά την υλοποίηση της εργασίας στη Β φάση, δημιουργήθηκαν οι παρακάτω υποκλάσεις της Exception και χρησιμοποιούνται στην εφαρμογή μέσω της κλάσης Exceptions του πακέτου.

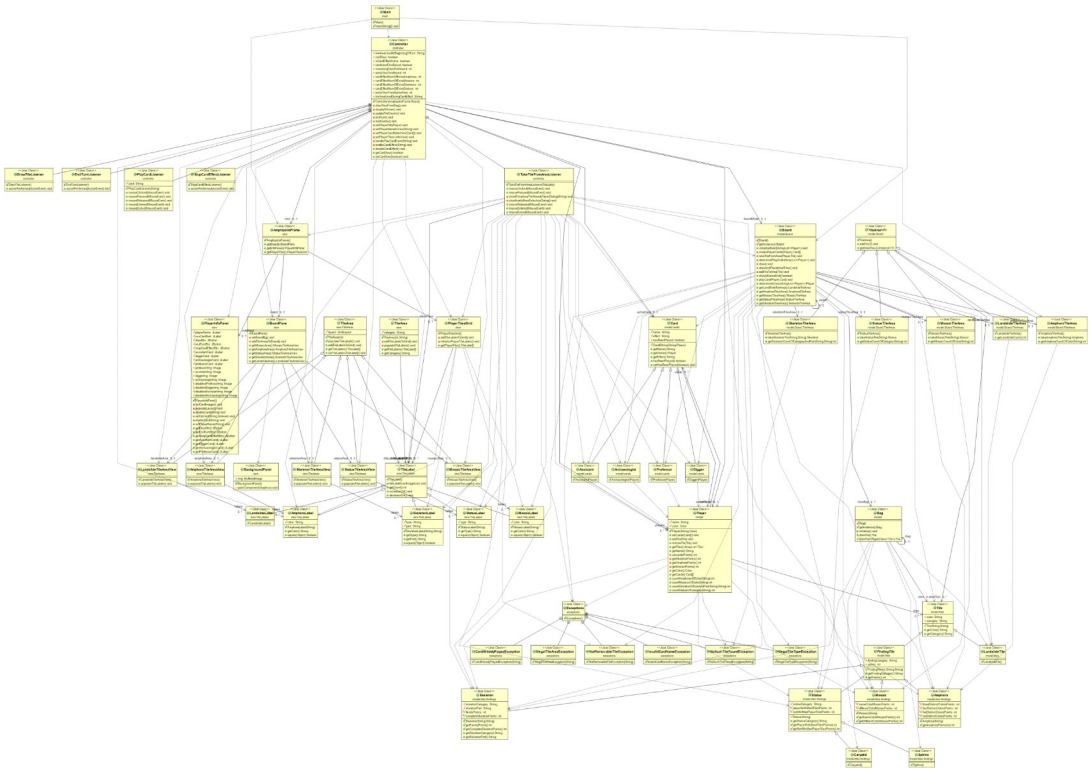
Custom Exceptions:

public static class IllegalTileTypeException(); // when an illegal tile type is passed to a method expecting a tile subclass

public static class NoSuchTileFoundException(); // when a method that works with the tile model data can't find a given tile or a tile with the given attributes

public static class NotRemovableTileException(); // When an attempt is made at removing a landslide tile from the area of the model

6.1.5 Το διάγραμμα UML του Project



6.2 Η τελική σχεδίαση της εφαρμογής και η εμπειρία χρήστη (UX)

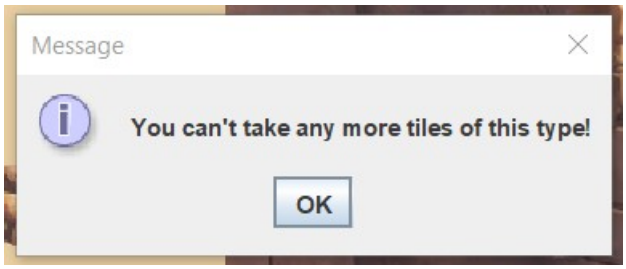
Παραπάνω φαίνεται μια εικόνα της τελικής γραφικής διεπαφής της εφαρμογής μόλις έχοντας ξεκινήσει το παιχνίδι.

Αρχικά, η κλάση AmphipolisFrame περιέχει όλα τα στοιχεία της διεπαφής και τα τοποθετεί βάση ενός BorderLayout.

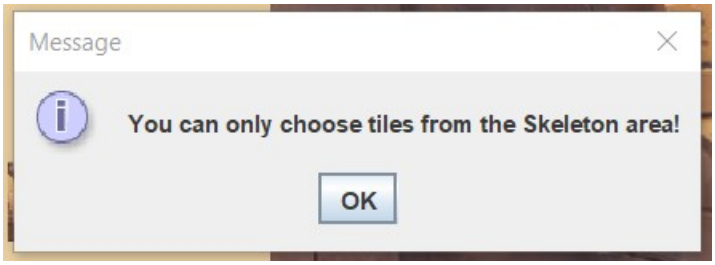
Στη συνέχεια, η κλάση BoardPane περιέχει τις διάφορες περιοχές πλακιδιών του ταμπλό και οι οποίες τοποθετήθηκαν στις κατάλληλες θέσεις “χειροκίνητα” στο JLayeredPane.

Κατά την σχεδίαση και υλοποίηση της διεπαφής επέλεξα η κάθε περιοχή να χρησιμοποιεί ένα GridLayout για την εμφάνιση των πλακιδίων. Έτσι, ο χρήστης βλέπει αμέσως όλα τα διαφορετικά πλακίδια που περιέχει η κάθε περιοχή. Ακόμα, ορίζοντας την κλάση TileLabel έθεσα έναν μετρητή στα δεξιά του κάθε πλακιδίου έτσι ώστε ο χρήστης να βλέπει εύκολα πόσα πλακίδια κάθε τύπου υπάρχουν σε μια δεδομένη στιγμή στο ταμπλό.

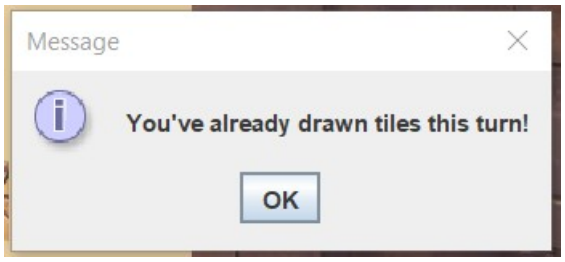
Κάθε πλακίδιο του ταμπλό/των περιοχών είναι αλληλεπιδράσιμο απλώς πατώντας πάνω του για να το τραβήξει ο παίχτης. Στην περίπτωση όπου παραβιάζεται κάποιος κανόνας (π.χ. προσπαθεί ένας παίχτης να τραβήξει τρία πλακίδια στη σειρά του), ή τελειώνει το παιχνίδι κλπ, γίνεται χρήση JOptionPane dialogs για την ενημέρωση του χρήστη όπως βλέπετε παρακάτω στις εικόνες για μερικές από τις διάφορες περιπτώσεις εμφάνισής διαλόγων.



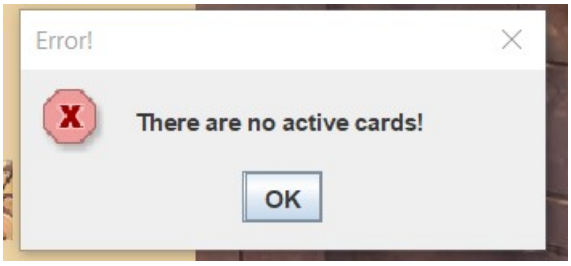
Διάλογος που εμφανίζεται όταν ο χρήστης δεν μπορεί να πάρει άλλα πλακίδια τέτοιου τύπου (π.χ. έχουν τελειώσει).



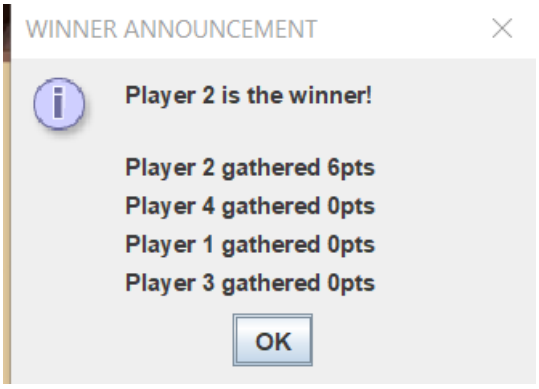
Διάλογος που εμφανίζεται όταν επιλέξει ο παίχτης να πάρει πλακίδιο από μια περιοχή διαφορετική από αυτήν που επέλεξε στην αρχή της σειράς του (εδω SkeletonArea).



Διάλογος που εμφανίζεται όταν επιλέξει ο χρήστης να τραβήξει ξανά πλακίδια από την σακούλα.



Διάλογος που εμφανίζεται όταν ο χρήστης πατάει στο “Stop card effect” κουμπί χωρίς να έχει παίξει κάποια κάρτα.



Διάλογος που εμφανίζεται στο τέλος του παιχνιδιού.

Όσον αφορά το δεξί κομμάτι της διεπαφής (PlayerInfoPanel) , εδώ έχουν χρησιμοποιηθεί Jlabels για τις κάρτες και στις οποίες έχει ανατεθεί ένα ActionListener έτσι ώστε όταν πατιούνται να μπορούμε να χειριστούμε το γεγονός.

Για την στοίχιση των διαφόρων κομματιών της διεπαφής έχει χρησιμοποιηθεί ένα GridBagLayout με τα κατάλληλα GridBagConstraints για το κάθε κομμάτι ξεχωριστά ώστε να μπορούν να στοιχιθούν με όμορφο και διαφορετικό τρόπο.

Τέλος, το κάτω τμήμα της διεπαφής αποτελεί την συλλογή πλακιδίων του εκάστοτε παίχτη. Αυτή περιέχει ένα TileLabel για όλα τα δυνατά πλακίδια που μπορεί να έχει ο παίχτης και καθώς αλληλεπιδρά με το ταμπλό και παίρνει πλακίδια, οι μετρητές του κατάλληλου πλακιδίου στη συλλογή του ανανεώνονται. Η διάταξη τους έγινε μέσω ενός GirdLayout για ευκολία και εδώ.

6.3 Σχεδιαστικές και προγραμματιστικές αποφάσεις και η επιρροή τους στον χρήστη

Η σχεδίαση της εφαρμογής όπως αναλύθηκε και παραπάνω, έγινε με τέτοιο τρόπο ώστε το παιχνίδι να είναι εύκολο και ξεκάθαρο στον χρήστη. Όλες οι πληροφορίες που χρειάζεται είναι άμεσα προσβάσιμες και ο στόχος του κάθε τμήματος του παιχνιδιού είναι ξεκάθαρος.

Η μοναδική σχεδιαστική αλλά και προγραμματιστική απόφαση που ίσως να επηρεάζει την εμπειρία του χρήστη είναι η δυνατότητα να ακυρωθεί νωρίς η ιδιότητα μιας κάρτας μέσω του κουμπιού “Stop card effect”. Ο λόγος γι’αυτό είναι πως τώρα ο χρήστης έχει να χειριστεί ένα ακόμα κουμπί το οποίο όμως προσφέρει μεγαλύτερη ευελιξία στον τρόπο παιχνιδιού του κάθε παίχτη.

Πιο αναλυτικά, αρχικά είχα υλοποιήσει την εργασία με τέτοιο τρόπο ώστε από την στιγμή που θα παιχτεί μια κάρτα, ο παίχτης θα πρέπει να τραβήξει όλα τα πλακίδια που του επιτρέπει η ιδιότητα της κάρτας προτού μπορεί να συνεχίσει να τραβάει πλακίδια ή να τελειώσει τη σειρά του. Για παράδειγμα, αν ένας παίχτης έπαιζε μια κάρτα Digger αφότου διάλεγε μια περιοχή και τράβαγε ένα πλακίδιο, θα έπρεπε να τραβήξει πρώτα τα 2 πλακίδια που του προσφέρει η κάρτα προτού μπορεί να τελειώσει τη σειρά του ή να τραβήξει το δεύτερό πλακίδιο της σειράς του. Αυτό όμως δημιουργούσε και αρνητική εμπειρία στον χρήστη και δεν εκτελούσε επαρκώς τις ιδιότητες καρτών που επέτρεπαν ΕΩΣ και 2 για παράδειγμα πλακίδια επιπλέον μιας και έπρεπε να πάρει και τα δυο.

Με την σχεδίαση της Β φάσης και την υλοποίηση που επέλεξα, το πρόβλημα αυτό λύνεται καθώς πατώντας το κουμπί αφότου έχει παιχτεί μια κάρτα, ο παίχτης μπορεί να σταματήσει την ιδιότητα της κάρτας όποτε επιθυμεί και έτσι είτε να επιλέξει μερικά μόνο πλακίδια από την ιδιότητα αλλά και να συνεχίσει να τραβάει πλακίδια που του έχουν μείνει για την σειρά του.

Προσωπικά, θεωρώ πως αυτή η σχεδιαστική απόφαση και υλοποίηση είναι μακράν καλύτερη της αρχικής και έχει θετική επιρροή στον τελικό χρήστη.

6.4 Προβλήματα που αντιμετωπίστηκαν

Το μοναδικό πρόβλημα που αντιμετώπισα κατά την υλοποίηση της εργασίας ήταν η εύρεση του τρόπου με τον οποίο το παιχνίδι θα εφάρμοζε τις ιδιότητες των διαφόρων καρτών χαρακτήρων.

Συγκεκριμένα, πως θα γνώριζα πόσα πλακίδια και ποιού τύπου μπορούν, πρέπει ή έχουν ήδη τραβηχθεί και από ποια περιοχή, πότε ο παίχτης τραβάει πλακίδια από την περιοχή του και μετρούν ως προς τα 2 πλακίδια ανα σειρά και πότε προς τα επιπλέον πλακίδια που επιτρέπει η ιδιότητα μιας κάρτας κλπ.

Μετά από αρκετές δοκιμές και σκέψη, κατέληξα στο να χρησιμοποιήσω τον παρακάτω αλγόριθμο ο οποίος ελέγχει αν κάποια κάρτα είναι ενεργή ή όχι και αναλόγως χειρίζεται το γεγονός με βάση τους κανόνες για την σειρά του παίχτη ή βάση τους κανόνες της εκάστοτε κάρτας. Ο παρακάτω αλγόριθμος περιέχεται μέσα στην μέθοδο onMousePressed της εσωτερικής κλάσης της κλάσης Controller που τρέχει όποτε ένας παίχτης επιλέγει ένα πλακίδιο από το ταμπλό.

Σημαντικό: Για ευκολότερη ανάγνωση προτείνω να ανοιχτεί η κλάση Controller.java σε κάποιον text editor/IDE και να πλοηγηθείτε στη γραμμή 517. Μια σύντομη περιγραφή του αλγορίθμου ακολουθεί μετά τον κώδικα.

Αλγόριθμος

```
        int totalTilesRemainingThisRound = remainingTilesThisRound + extraTilesThisRound
;


       if (clickedTile.getCount() > 0 && totalTilesRemainingThisRound > 0) {
         Optional<TileLabel> optionalTileLabel;
         TileLabel[] playerTiles = view.getPlayerTiles().getPlayerTiles();

         // Figure out which tile label was clicked in order to decrease its count and
also update the model

         if (clickedTile instanceof AmphoraLabel) {
           // update selected tile area the first time an area is selected
           if (tileAreaUsedAtBeginningOfTurn.isEmpty()) {
             tileAreaUsedAtBeginningOfTurn = "Amphora";
             remainingTilesThisRound--;
           }
           else { // player has already selected a tile area
             if (!tileAreaUsedAtBeginningOfTurn.equals("Amphora") && !
isCardEffectActive) { // show msg and don't complete action
               showInvalidAreaSelectionDialog();
               return;
             }

             if (isCardEffectActive) {
               if (!tileAreaUsedAtBeginningOfTurn.equals("Amphora")) { // find out whi
ch card has been used and act accordingly
                 if (activeCard instanceof Professor) {
                   if (cardEffectNumOfExtraAmphoras > 0) {
                     cardEffectNumOfExtraAmphoras--;
                     extraTilesThisRound--;
                     if (extraTilesThisRound == 0) isCardEffectActive = false;
                   } else {
                     showExtraAreaTileAlreadyTakenDialog("amphora");
                     return; // no more amphoras should be drawn due to the effect
                   }
                 } else if (activeCard instanceof Archaeologist) { // if the player ch
ose to draw tiles from this area using the effect or if he's drawing for the first tim
e after playing the card
                   if(extraTilesFromSameArea > 0) {
                     extraTilesFromSameArea--;
                     extraTilesThisRound--;
                     if(extraTilesFromSameArea == 0) isCardEffectActive = false;
                   }
                 } else if(activeCard instanceof Digger) { // digger is active but the
player selected a tile from a forbidden area
                   JOptionPane.showMessageDialog(null,"You can only pick extra tiles f
rom the " + tileAreaUsedAtBeginningOfTurn +
                       " area while playing the Digger card!");
                   return;
                 }
               } else { // tileAreaUsed == "amphora' and card effect is active
```

```
                if (activeCard instanceof Digger) {
                  if (cardEffectNumOfExtraAmphoras > 0) {
                    cardEffectNumOfExtraAmphoras--;
                    extraTilesThisRound--;
                    if (cardEffectNumOfExtraAmphoras == 0) isCardEffectActive = false
;
                  }
                } else {
                  JOptionPane.showMessageDialog(null,"Can't take tile from this area
when playing the " + activeCard.getName() + " card!");
                  return;
                }
              }
              if (activeCard instanceof Assistant && extraTilesFromSameArea > 0) {
                extraTilesFromSameArea--;
                extraTilesThisRound--;
                if (extraTilesFromSameArea == 0) isCardEffectActive = false;
              }
            } else {
              remainingTilesThisRound--;
            }
          }

          optionalTileLabel = Arrays.stream(playerTiles).filter(tileLabel ->
              tileLabel instanceof AmphoraLabel).
              filter(tileLabel -> tileLabel.equals(clickedTile)).findFirst();


          try {
            boardModel.takeTileFromArea(activePlayer, new Amphora(((AmphoraLabel) opti
onalTileLabel.get()).getColor()));
          } catch (Exceptions.NotRemovableTileException | Exceptions.NoSuchTileFoundEx
ception notRemovableTileException) {
            notRemovableTileException.printStackTrace();
          }

          if (optionalTileLabel.isPresent()) {
            optionalTileLabel.get().increaseCnt();
          }

        } else if (clickedTile instanceof MosaicLabel) {
          // update selected tile area the first time an area is selected
          if (tileAreaUsedAtBeginningOfTurn.isEmpty()) {
            tileAreaUsedAtBeginningOfTurn = "Mosaic";
            remainingTilesThisRound--;
          }
          else { // player has selected a tile area
            if (!tileAreaUsedAtBeginningOfTurn.equals("Mosaic") && !
isCardEffectActive) { // show msg and don't complete action
              showInvalidAreaSelectionDialog();
              return;
            }

            if (isCardEffectActive) {
              if (!tileAreaUsedAtBeginningOfTurn.equals("Mosaic")) { // find out whic
h card has been used and act accordingly
```

```
                if (activeCard instanceof Professor) {
                  if (cardEffectNumOfExtraMosaics > 0) {
                    cardEffectNumOfExtraMosaics--;
                    extraTilesThisRound--;
                    if (extraTilesThisRound == 0) isCardEffectActive = false;
                  }  else {
                    showExtraAreaTileAlreadyTakenDialog("mosaic");
                    return; // no more mosaics should be drawn due to the effect
                  }
                } else if (activeCard instanceof Archaeologist) { // if the player ch
ose to draw tiles from this area using the effect or if he's drawing for the first tim
e after playing the card
                  if(extraTilesFromSameArea > 0) {
                    extraTilesFromSameArea--;
                    extraTilesThisRound--;
                    if(extraTilesFromSameArea == 0) isCardEffectActive = false;
                  }
                } else if(activeCard instanceof Digger) { // digger is active but the
player selected a tile from a forbidden area
                  JOptionPane.showMessageDialog(null,"You can only pick extra tiles f
rom the " + tileAreaUsedAtBeginningOfTurn +
                      " area while playing the Digger card!");
                  return;
                }
              } else { // tileAreaUsed == "mosaic' and card effect is active
                if (activeCard instanceof Digger) {
                  if (cardEffectNumOfExtraMosaics > 0) {
                    cardEffectNumOfExtraMosaics--;
                    extraTilesThisRound--;
                    if (cardEffectNumOfExtraMosaics == 0) isCardEffectActive = false;
                  }
                } else {
                  JOptionPane.showMessageDialog(null,"Can't take tile from this area
when playing the " + activeCard.getName() + " card!");
                  return;
                }
              }
              if (activeCard instanceof Assistant && extraTilesFromSameArea > 0) {
                extraTilesFromSameArea--;
                extraTilesThisRound--;
                if (extraTilesFromSameArea == 0) isCardEffectActive = false;
              }
            } else {
              remainingTilesThisRound--;
            }
          }

          optionalTileLabel = Arrays.stream(playerTiles).filter(tileLabel ->
              tileLabel instanceof MosaicLabel).
              filter(tileLabel -> tileLabel.equals(clickedTile)).findFirst();


          try {
            boardModel.takeTileFromArea(activePlayer, new Mosaic(((MosaicLabel) option
alTileLabel.get()).getColor()));
```

```
            } catch (Exceptions.NotRemovableTileException | Exceptions.NoSuchTileFoundEx
ception notRemovableTileException) {
              notRemovableTileException.printStackTrace();
            }

            if (optionalTileLabel.isPresent()) {
              optionalTileLabel.get().increaseCnt();
            }

          } else if (clickedTile instanceof SkeletonLabel) {
            // update selected tile area the first time an area is selected
            if (tileAreaUsedAtBeginningOfTurn.isEmpty()) {
              tileAreaUsedAtBeginningOfTurn = "Skeleton";
              remainingTilesThisRound--;
            }
            else { // player has selected a tile area
              if (!tileAreaUsedAtBeginningOfTurn.equals("Skeleton") && !
isCardEffectActive) { // show msg and don't complete action
                showInvalidAreaSelectionDialog();
                return;
              }

              if (isCardEffectActive) {
                if (!tileAreaUsedAtBeginningOfTurn.equals("Skeleton")) { // find out wh
ich card has been used and act accordingly
                  if (activeCard instanceof Professor) {
                    if (cardEffectNumOfExtraSkeletons > 0) {
                      cardEffectNumOfExtraSkeletons--;
                      extraTilesThisRound--;
                      if (extraTilesThisRound == 0) isCardEffectActive = false;
                    }  else {
                      showExtraAreaTileAlreadyTakenDialog("skeleton");
                      return; // no more skeletons should be drawn due to the effect
                    }
                  } else if (activeCard instanceof Archaeologist) { // if the player ch
ose to draw tiles from this area using the effect or if he's drawing for the first tim
e after playing the card
                    if(extraTilesFromSameArea > 0) {
                      extraTilesFromSameArea--;
                      extraTilesThisRound--;
                      if(extraTilesFromSameArea == 0) isCardEffectActive = false;
                    }
                  } else if(activeCard instanceof Digger) { // digger is active but the
player selected a tile from a forbidden area
                    JOptionPane.showMessageDialog(null,"You can only pick extra tiles f
rom the " + tileAreaUsedAtBeginningOfTurn +
                        " area while playing the Digger card!");
                    return;
                  }
                } else { // tileAreaUsed == "skeleton" and card effect is active
                  if (activeCard instanceof Digger) {
                    if (cardEffectNumOfExtraSkeletons > 0) {
                      cardEffectNumOfExtraSkeletons--;
                      extraTilesThisRound--;
                      if (cardEffectNumOfExtraSkeletons == 0) isCardEffectActive = fals
e;
```

```
                    }
                } else {
                    JOptionPane.showMessageDialog(null,"Can't take tile from this area
when playing the " + activeCard.getName() + " card!");
                    return;
                }
            }
            if (activeCard instanceof Assistant && extraTilesFromSameArea > 0) {
                extraTilesFromSameArea--;
                extraTilesThisRound--;
                if (extraTilesFromSameArea == 0) isCardEffectActive = false;
            }
        } else {
            remainingTilesThisRound--;
        }
    }

    optionalTileLabel = Arrays.stream(playerTiles).filter(tileLabel ->
        tileLabel instanceof SkeletonLabel).
        filter(tileLabel -> tileLabel.equals(clickedTile))
        .findFirst();


    try {
        SkeletonLabel skeleton = (SkeletonLabel) optionalTileLabel.get();
        boardModel.takeTileFromArea(activePlayer, new Skeleton(skeleton.getType(),
skeleton.getPart()));
    } catch (Exceptions.NotRemovableTileException | Exceptions.NoSuchTileFoundEx
ception notRemovableTileException) {
        notRemovableTileException.printStackTrace();
    }

    if (optionalTileLabel.isPresent()) {
        optionalTileLabel.get().increaseCnt();
    }

} else if (clickedTile instanceof StatueLabel) {
    // update selected tile area the first time an area is selected
    if (tileAreaUsedAtBeginningOfTurn.isEmpty()) {
        tileAreaUsedAtBeginningOfTurn = "Statue";
        remainingTilesThisRound--;
    }
    else { // player has selected a tile area
        if (!tileAreaUsedAtBeginningOfTurn.equals("Statue") && !
isCardEffectActive) { // show msg and don't complete action if it doesn't match the pr
ev selected tile area
            showInvalidAreaSelectionDialog();
            return;
        }

        if (isCardEffectActive) {
            if (!tileAreaUsedAtBeginningOfTurn.equals("Statue")) { // find out whic
h card has been used and act accordingly
                if (activeCard instanceof Professor) {
                    if (cardEffectNumOfExtraStatues > 0) {
                        cardEffectNumOfExtraStatues--;
```

```
                    extraTilesThisRound--;
                    if (extraTilesThisRound == 0) isCardEffectActive = false;
                  }  else {
                    showExtraAreaTileAlreadyTakenDialog("statue");
                    return; // no more statues should be drawn due to the effect
                  }
                } else if (activeCard instanceof Archaeologist) { // if the player ch
ose to draw tiles from this area using the effect or if he's drawing for the first tim
e after playing the card
                  if(extraTilesFromSameArea > 0) {
                    extraTilesFromSameArea--;
                    extraTilesThisRound--;
                    if(extraTilesFromSameArea == 0) isCardEffectActive = false;
                  }
                } else if(activeCard instanceof Digger) { // digger is active but the
player selected a tile from a forbidden area
                  JOptionPane.showMessageDialog(null,"You can only pick extra tiles f
rom the " + tileAreaUsedAtBeginningOfTurn +
                      " area while playing the Digger card!");
                  return;
                }
              } else { // tileAreaUsed == "statue" and card effect is active
                if (activeCard instanceof Digger) {
                  if (cardEffectNumOfExtraStatues > 0) {
                    cardEffectNumOfExtraStatues--;
                    extraTilesThisRound--;
                    if (cardEffectNumOfExtraStatues == 0) isCardEffectActive = false;
                  }
                } else {
                  JOptionPane.showMessageDialog(null,"Can't take tile from this area
when playing the " + activeCard.getName() + " card!");
                  return;
                }
              }
              if (activeCard instanceof Assistant && extraTilesFromSameArea > 0) {
                extraTilesFromSameArea--;
                extraTilesThisRound--;
                if (extraTilesFromSameArea == 0) isCardEffectActive = false;
              }
            } else {
              remainingTilesThisRound--;
            }
          }

          optionalTileLabel = Arrays.stream(playerTiles).filter(tileLabel ->
              tileLabel instanceof StatueLabel).
              filter(tileLabel -> tileLabel.equals(clickedTile)).findFirst();


          StatueLabel statueLbl = (StatueLabel) optionalTileLabel.get();
          Statue statue;
          try {
            statue = (Statue) Class.forName("com.csd4020.model.tiles.findings." + stat
ueLbl.getType()).newInstance();
            boardModel.takeTileFromArea(activePlayer, statue);
```

```
            } catch (Exceptions.NotRemovableTileException | Exceptions.NoSuchTileFoundEx
ception | IllegalAccessException | InstantiationException | ClassNotFoundException not
RemovableTileException) {
                notRemovableTileException.printStackTrace();
            }

            if (optionalTileLabel.isPresent()) {
                optionalTileLabel.get().increaseCnt();
            }

        } else throw new IllegalArgumentException("Clicked tile isn't valid!");
      } else if (clickedTile.getCount() == 0) {
        JOptionPane.showMessageDialog(null, "You can't take any more tiles of this typ
e!");
      } else {
        JOptionPane.showMessageDialog(null, "You can't take any more tiles this turn!"
);
      }
```

Επεξήγηση:

Αρχικά ο αλγόριθμος υπολογίζει πόσα πλακίδια μπορεί να επιλέξει ο παίχτης στη σειρά του κάθε φορά που προσπαθεί να τραβήξει ένα πλακίδιο από μια περιοχή του ταμπλό. Στη συνέχεια, εφόσον το totalTilesRemainingThisRound > 0 και υπάρχουν πλακίδια του τύπου που επέλεξε, ελέγχει τι είδος πλακιδίου επέλεξε να πάρει ο χρήστης μέσω του instanceof τελεστή. Εφόσον βρεθεί ο τύπος, ελέγχει αν είναι η πρώτη φορά που επιλέγει πλακίδιο αυτή τη σειρά έτσι ώστε αν είναι να θέσει την αντίστοιχη περιοχή ως την επιλεγμένη περιοχή. Σε αυτήν την περίπτωση, βρίσκει το αντίστοιχο πλακίδιο και το δίνει στον παίχτη ενημερώνοντας το view και το model. Επίσης μειώνει το remainingTilesThisRound κατά ένα. Στην άλλη περίπτωση, ελέγχει αν δεν έχει παιχτεί κάρτα και ο παίχτης προσπαθεί να πάρει πλακίδιο από περιοχή που δεν επέλεξε στην αρχή. Αν αυτό ισχύει δείχνει κατάλληλο μήνυμα και σταματά. Διαφορετικά, ελέγχει αν έχει παιχτεί κάρτα, αν δεν έχει παιχτεί, ακολουθεί τα ίδια βήματα που αναφέρθηκαν προ ολίγου ώστε να δώσει το πλακίδιο. Στην περίπτωση όπου έχει παιχτεί κάρτα, ελέγχει τις κατάλληλες και διάφορες συνθήκες για να βρει ποια απ'όλες τις κάρτες παίχτηκε. Μόλις καταλήξει και εάν όλες η συνθήκες της εκάστοτε κάρτας ικανοποιούνται, εκτελεί τις κατάλληλες ενέργειες βάση των ιδιοτήτων της κάρτας, δίνει το πλακίδιο στον παίχτη και τερματίζει μειώνοντας φυσικά και τα κατάλληλα πεδία ώστε να μην ξανατρέξει ο αλγόριθμος όταν δεν θα δικαιούται περισσότερα πλακίδια ο παίχτης.

6.5 Επιπλέον Πληροφορίες

Αξίζει να αναφερθεί πως λόγω κακής προσωπικής διαχείρισης χρόνου, επέλεξα να μην γράψω JUnit Tests για τον έλεγχο της ορθότητας του προγράμματος. Παρ'όλο που δεν αντιμετώπισα κάποιο ιδιαίτερο πρόβλημα με την υλοποίηση της εφαρμογής (εκτός αυτού που αναφέρεται στην ενότητα 6.4), το σωστό θα ήταν να έχω γράψει tests.

7. Συμπεράσματα

Σε αυτήν την ενότητα, θα αναφερθώ σε κάποιες παρατηρήσεις μου για την εργασία και την χρήση αρχιτεκτονικής κώδικα, τυχόν προβλήματα που αντιμετώπισα , καθώς και κάποιες πιθανές “παγίδες” που εντόπισα κατά την διάρκεια της ανάπτυξης της.

Αρχικά , κατά την διάρκεια της έρευνας μου για την αρχιτεκτονική MVC και ιδιαίτερα κατά την προσπάθεια υλοποίησης της, γρήγορα συνειδητοποίησα πως η ανάπτυξη κώδικα γίνεται ευκολότερη μιας και ο κώδικας χωρίζεται σε ξεχωριστά πακέτα με σαφείς ρόλους το καθένα. Επίσης, η εξσφαλμάτωση γίνεται απλούστερη καθώς κάθε μονάδα της εφαρμογής μπορεί να ελεγχθεί μεμονωμένα εφόσον έχουμε κατάλληλη διαχώριση κώδικα σε “επίπεδα”. Ακόμα, η τήρηση μιας τέτοιας αρχιτεκτονικής κώδικα σε μια αρκετά μεγαλύτερη εφαρμογή, είναι εύκολο να φανταστεί κανείς πώς θα έκανε την συνεργασία και την ανάπτυξη πιο γρήγορη και εύκολη προσφέροντας μας έτσι, ευκολότερη επεκτασιμότητα (scalability) και συντηρησιμότητα (maintainability) του κώδικα.

Στη συνέχεια, μια δυσκολία που αντιμετώπισα ήταν να σκεφτώ πως θα αναπαρασταθούν τα διάφορα δεδομένα και η γραφική διεπαφή του παιχνιδιού μέσω κλάσεων και μεθόδων, χωρίς την απευθείας (ολόκληρη) υλοποίηση τους όμως. Το θετικό της ανάλυσης του προβλήματος σε υποπροβλήματα, κλάσεις και αλγορίθμους είναι πως με βοήθησε στην αποφυγή λαθών που ίσως να δημιουργούνταν αν ξεκινούσα να γράφω κώδικα αμέσως χωρίς μελέτη (π.χ. μη επαρκής χρήση κληρονομικότητας και διεπαφών, έλλειψη διαχωρισμού του κώδικα σε αυτόνομες μονάδες, κλπ).

Τέλος, κάτι που προσωπικά θεωρώ “παγίδα” στην υλοποίηση της εργασίας με την αρχιτεκτονική MVC και που με προβλημάτισε αρχικά, ήταν ο τρόπος αλληλεπίδρασης του χρήστη με τη γραφική διεπαφή. Η πρώτη μου σκέψη ήταν μέσα στις ίδιες τις κλάσης του πακέτου View , να ορίσω και να θέσω κατάλληλους Listeners οι οποίοι θα ήταν υπεύθυνοι για την αλληλεπίδραση του χρήστη με τη διεπαφή. Ύστερα από σκέψη όμως, συνειδητοποίησα πως κάτι τέτοιο θα καταργούσε την αρχιτεκτονική MVC και πως αυτό είναι δουλεία του Controller μιας και είναι ο εγκέφαλος του παιχνιδιού και διότι θέλουμε με την αλληλεπίδραση του χρήστη, να κάνουμε αλλαγές και στο UI και στο model του παιχνιδιού.

Φάση Β:

Επιπρόσθετα, θα ήθελα να αναφέρω πόσο χρήσιμη είναι η αρχική σχεδίαση της εφαρμογής και οι διάφοροι έλεγχοι και σκέψεις που γίνονται κατά την διαδικασία αυτή, για την υλοποίηση της αργότερα. Έχοντας κάνει την αρχική σχεδίαση, η υλοποίηση έγινε γρήγορα και εύκολα ομολογώ καθώς το πρόβλημα της δημιουργίας της είχε ήδη διαχωρισθεί σε υπό-προβλήματα και έτσι η

επίλυση τους ήταν ευκολότερη με αποτέλεσμα να επιλυθεί και το αρχικό πρόβλημα. Η τήρηση της αρχιτεκτονικής έκανε επίσης τα πράγματα πιο ξεκάθαρα και βοήθησε στην ταχύτερη ανάπτυξη και συντήρηση του κώδικα παρά τις όσες αλλαγές χρειάστηκαν να γίνουν κατά την Β φάση.