Πανεπιστήμιο Κρήτης –Τμήμα Επιστήμης Υπολογιστών

ΗΥ252– Αντικειμενοστρεφής Προγραμματισμός

Διδάσκων: Ι. Τζίτζικας

Χειμερινό Εξάμηνο 2024-2025

ΑΝΑΖΗΤΩΝΤΑΣ ΤΑ ΧΑΜΕΝΑ ΜΙΝΩΙΚΑ ΑΝΑΚΤΟΡΑ

Elias Rishmawi
csd5299
8/1/2025

Περιεχόμενα

1. Εισαγωγή

In order to do this game we used the MVC model which will allow us to implement the game by dividing it into 3 parts, which are the model which has all the data of the game, the view which allow us to see the game and interact with it by using buttons…, and the controller which will be the connecting link between the model and the view, and the mind of the game. In this report we will talk about these parts in detail.

2. Η Σχεδίαση και οι Κλάσεις του Πακέτου Model

This package will contain all the data of the game. We divided it into multiple packages:

card package: which has the interface Card and the classes who implement the Card interface which are: SimpleCard class, Ariadni class, Minotavro class. Also, it contains the CardStack interface and the classes that implements it which are: CardDeck and DiscardDeck.

findings package: which has the interface Findings and the classes that implement it which are: RareFinds, SimpleFinds, Murals. path package : which has Path class, Board class, Position class, and an enum PathColor.

pawn package: which has the Pawn class and its subclasses: Archaeologist and Theseus class. player package: which has the Player class and an enum PlayerColor.

exceptions package: has an IllegalMoveException. (might add other exceptions in the future)

Card package

Card Interface and Other Classes for Cards

By first creating the Card interface, we are given the ability to access the data without having to define the category of the card.

This interface provides us with the following methods:

1. int getValue(); (Accessor)
   Returns the value of the card.
2. int getSteps(); (Accessor)
   Returns the steps of the card. (how many steps does the card move the pawn)
3. void setSteps(int steps); (Transformer) Sets the steps of the card
4. Path getPath(); (Accessor)
   Returns the path of the card.
5. String getImagePath(); (Accessor) Returns the path image of the card

6. void play(Pawn pawn); changes the pawn's position depending on which card was played.

Now, we have the classes that implements the Card interface:

Class SimpleCard

This class represents the Simple card. Each card has a value, and a path, and it moves the pawn 1 step forward. Here, we will list all the attributes and other methods that this card has. (except for those it implements through the Card interface)

The attributes:

1. private final int value; (the value of the card)
2. private final int steps; (the steps of the card = 1)
3. private final Path path; (the path of the card)
4. private final String imagePath; (the image's path of the card)

The other Methods:

1. boolean matchCard(Card prvCard);
   Returns true if it is legal to thow this card after the prvCard, false otherwise.

Class Ariadni

This class represents the Ariadni card. It doesn't have a value, it moves the pawn 2 steps forward. Here we will list the attributes that this class has (all its method are overriding the Card interface)

The attributes:

1. private int steps; (the steps of the card = 2)
2. private final Path path; (the path of the card)
3. private final String imagePath; (the image's path of the card)

Class Minotavro

This class represents the Minotavro card. It doesn't have a value. It moves the opponent pawn 2 steps backwards if it is Archaeologist, or doesn't let the opponent play for a round if the pawn is Theseus. Here we will list the attributes that this class has
(all its method are overriding the Card interface)

The attributes:

1. private int steps; (the steps of the card = -2)
2. private final Path path; (the path of the card)
3. private final String imagePath; (the image's path of the card)

CardStack interface and other classes for CardStack

This interface provides us with the following methods:

1. void push(Card card); to put a card in the stack
2. boolean isEmpty (); (observer)
   Returns true if the stack is empty, False otherwise
3. void clear (); to clear the stack
4. int getSize();
   Returns the number of cards in the stack

Now we have the classes that implements the CardStack:

Class CardDeck

This class implements the CardStack interface. (a static implementation of a stack). Essentially it is an array with 100 positions that contains the cards of the game.

When a CardDeck is created it will be initialized and shuffled.

The attributes:

1. private final static int capacity = 100; (the size of the Deck)
2. private Card[] cards; (array of cards)
3. private int top; (the index of the top of the stack)
4. private List<SimpleCard> simpleCards; (list of simple cards)
5. private List<Ariadni> ariadniCards; (list of aridani cards)
6. private List<Minotavro> minotavroCards; (list of minotavro cards)

it overrides the methods from the CardStack interface, I will list bellow the other methods that it has

The Other methods:

1. public void createSimpleCards() creates 80 simple cards and puts it in the simple cards list
2. public void createAriadniCards() creates 12 ariadni cards and puts it in the ariadni cards list

3. public void createMinotavroCards() creates 8 minotavro cards and puts it in the minotavro cards list
4. public void initializeDeck() putting a 100 cards randomly from the 3 lists into the cards array (shuffle the deck)
5. public Card pop() throws IllegalStateException
   Returns the card that was drawed from the card deck, and removes it from the deck.

Class DiscardDeck

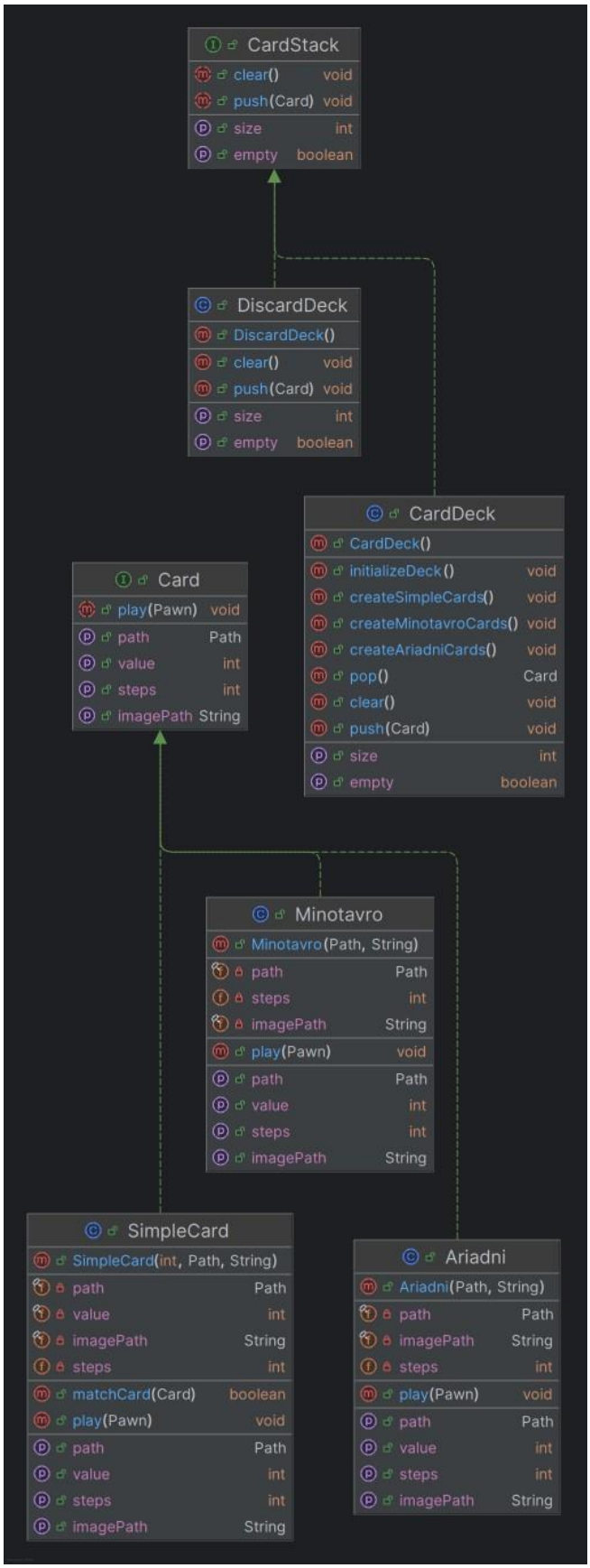
This class represents the discard deck and implements the CardStack interface. (dynamic implementation of a stack). Essentially it is an empty list and each time a player throws a card to it, it pushes this card.

The attributs:

1. private List<Card> cards; (list of cards)
2. private int top; (index to the top of the stack)

it overrides the methods from the CardStack interface, and it doesn't have any other methods.

This is the end of the card package. I will show a representation of the classes in this package via UML:


CardStack
clear() void
push(Card) void
size int
empty boolean
DiscardDeck
DiscardDeck()
clear() void
push(Card) void
size int
empty boolean
CardDeck
CardDeck()
initializeDeck() void
createSimpleCards() void
createMinotavroCards() void
createAriadniCards() void
pop() Card
clear() void
push(Card) void
size int
empty boolean
Card
play(Pawn) void
path Path
value int
steps int
imagePath String
Minotavro
Minotavro(Path, String)
path Path
steps int
imagePath String
play(Pawn) void
path Path
value int
steps int
imagePath String
SimpleCard
SimpleCard(int, Path, String)
path Path
value int
imagePath String
steps int
matchCard(Card) boolean
play(Pawn) void
path Path
value int
steps int
imagePath String
Ariadni
Ariadni(Path, String)
path Path
imagePath String
steps int
play(Pawn) void
path Path
value int
steps int
imagePath String

Findings package

Findings Interface and Other Classes for Findings

By first creating the Findings interface, we are given the ability to access the data without having to define the category of the Finding.

This interface provides us with the following methods:

1. int getPoints(); (Accessor)
   Returns the points that the finding has
2. String getImagePath(); (Accessor)
   Returns the image path of the finding

Now, we have the classes that implements the Findings interface:

Class RareFinds

Here, we will list all the attributes and other methods that this class has. (except for those it implements through the Findings interface).

The attributes:

1. private Path path; (the path of the finding)
2. private int points; (the points of the rare find
3. private String imagePath; (the image path of the finding)

Other Methods:

1. public Path getPath()
   Returns the path of the rare find

Class Murals

Here, we will list the attributes of this class. It doesn't have any additional methods other than the Findings interface.

The attributes:

1. private final int points; (the points of the mural)
2. private final String imagePath; (the image path)

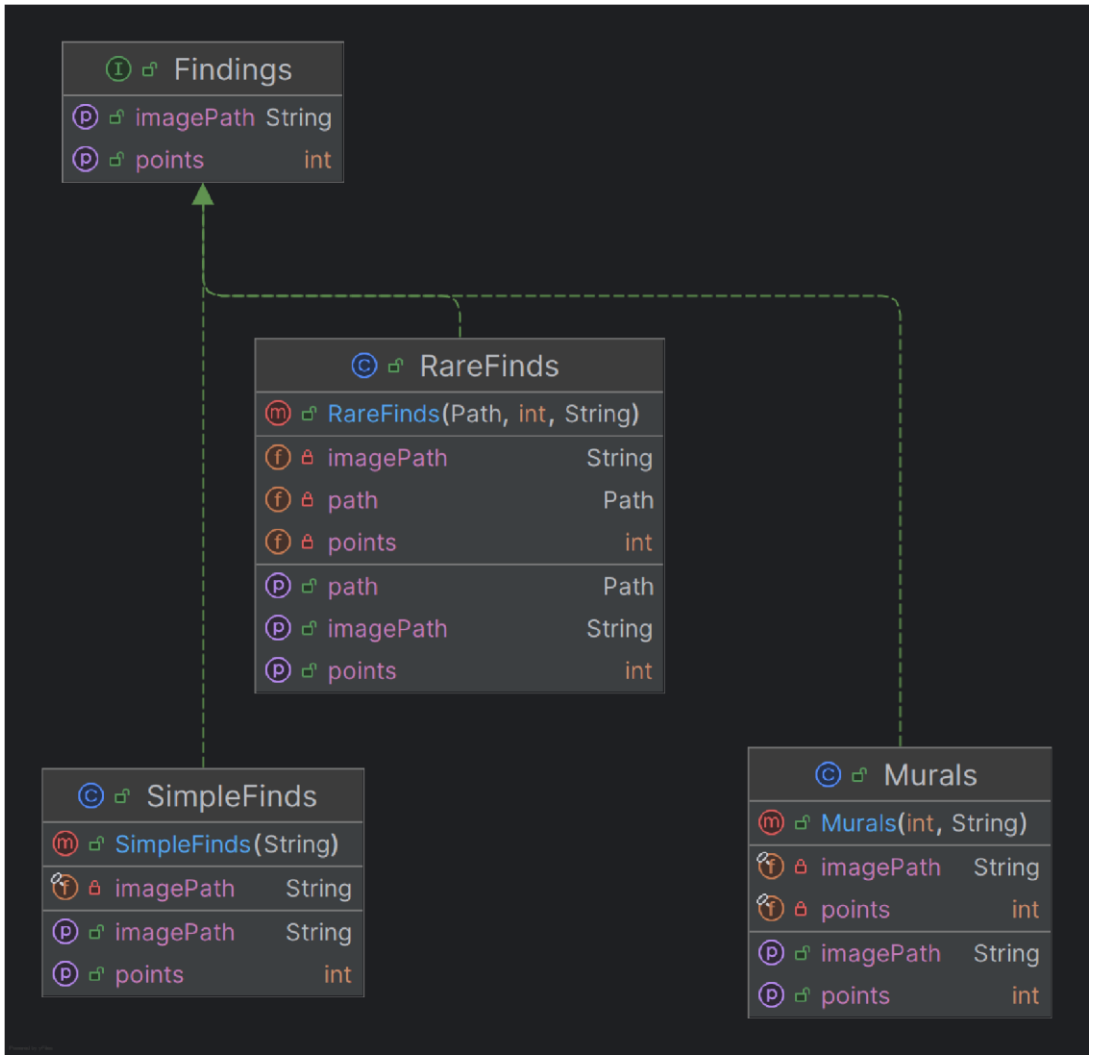
Class SimpleFinds

The simple find doesn't have points(points = 0). And doesn't have any additional methods

The attributes:

1. private final String imagePath; (the image's path)

here is a representation of the findings package in UML:



Path package

Class Path

This class represents the path. Essentially it is an array of 9 positions and it has a specific color.

The attributes:

1. private final String name; (each path has a name)
2. private final int lenght; (the length of the path, which is 9)
3. private final Position[] positions; (array of positions)
4. private final PathColor color; (each path has a color)

The methods:

1. public String getName() (Accessor) Returns the name of the path
2. public Position getPosition(int number) (Accessor) Returns the Position class depending on an index.
3. public PathColor getColor() (Accessor) Returns the color of the path
4. public void initializePath()
   Initializes the path with 9 positions that each has a different number, points, findings(if it has).

Class Position

This class represents the position. Essentially each position in the game has a number from 1 to 9, points, finding (not all have finding), and the color of the path.

The attributes:

1. private int number; (the number of the position)
2. private final int points; (the points of the position)
3. private Findings finds; (the finding in the position)
4. private final String imagePath; (image path of the position)
5. private final PathColor color; (the color of the position depending on its path)

The methods:

1. public boolean hasFinds() (Observer)
   Returns True if the position has a finding in it. False otherwise.
2. public void removeFinds() to remove the finding in the position if a certain scenario occurred (such as capturing a finding)
3. public int getPoints() (Accessor) Returns the points of the position
4. public Findings getFinds() (Accessor)
   Returns the finding in the position if it has one. Or NULL if it doesn't have a finding in the position.
5. public String getImagePath() (Accessor)

Returns the image's path (different image if a position has a finding)

6. public PathColor getColor() (Accessor) Returns the color of the position
7. public int getNumber() (Accessor) Returns the number of the position
8. public void setPosition(int number) (Transformer) Sets the number of the position

Class PathColor

It's an Enum class which contains the colors of the paths

(RED, YELLOW, WHITE, BLUE)

Class Board

This class represents the Board of the game. Essentially it has 4 paths (Knossos, malia, Phaistos, zakros) each path has different color.

The attributes:

1. private final Path knossos;
2. private final Path malia;
3. private final Path phaistos;
4. private final Path zakros;

The methods: (all are Accessors)

1. public Path getKnossos()
   Returns the first path (Knossos)
2. public Path getMalia()
   Returns the second path (Malia)
3. public Path getPhaistos()
   Returns the third path (Phaistos)
4. public Path getZakros()
   Returns the fourth path (Zakros)

Pawn package

Abstract Class Pawn

This class represents the pawns of the game. It is placed on a position on a specific path on the board, it has a color depending on the players. and it is either visible or invisible.

When a pawn is created it is placed on the first position. positionNum = 1, and it is invisible.

The attributes:

1. private int positionNum; (the position of the pawn (index))
2. private final Path path; (the path of the pawn)
3. private final String imagePath; (the image's path)
4. private final PlayerColor color; (the color of the pawn)
5. private boolean visible; (the visibility of the pawn)

The methods:

1. public int getPositionNum() (Accessor) Returns the position (index) that the pawn is in
2. public Position getPosition() (Accessor) Returns the Position that the pawn is in
3. public void setPosition(int position) (Transformer)
   Sets (updates) the position of the pawn in the game
4. public Path getPath() (Accessor) Returns the path of the pawn
5. public abstract void playedCard(Card card); abstract method lets the subclasses implement, that relies on the play() method in the Card interface, essentially it calls it when a card is played to change the position of the pawn
6. public String getImagePath() (Accessor) Returns the image path
7. public PlayerColor getColor() (Accessor)
   Returns the color of the pawn depending on the player
8. public boolean isVisible() (Observer)
   Returns true if the pawn is visible. False otherwise.
9. public void setAppear(boolean visible) (Transformer) Sets the visibility of the pawn.
10. public abstract void InteractWithBox(Player player, Position position); Abstract method that lets the subclasses implement, to make the essential changes when a pawn interact with a box. For example capturing a mural and destroying it.

The pawn class has 2 subclasses: Theseus class and Archaeologist class:

Class Theseus

This is a subclass of the Pawn class. It represents the Theseus. It extends the Pawn and uses the super command to gain access to the pawn class and initializes its color and

path. Also has another attribute which is destroyedBoxes which at first is initialized at 0, it represents the number of boxes that the theseaus destroyed, which will be used in the InteractWithBox method logic.

It overrides the abstract methods of the Pawn class : playedCard and InteractWithBox, also overrides the getImagePath().

The attributes:

1. private final String imagePath; (image path)
2. private int destroyedBoxes; (the numbers of boxes that the Theseus destroyed)

The Other Methods:

1. public int getDestroyedBoxes() (Accessor)
   Returns the number of boxes that the theseus destroyed
2. public void setDestroyedBoxes(int destroyedBoxes) (Transformer) Sets (updates) the number of boxes that is destroyed

Class Archaeologist

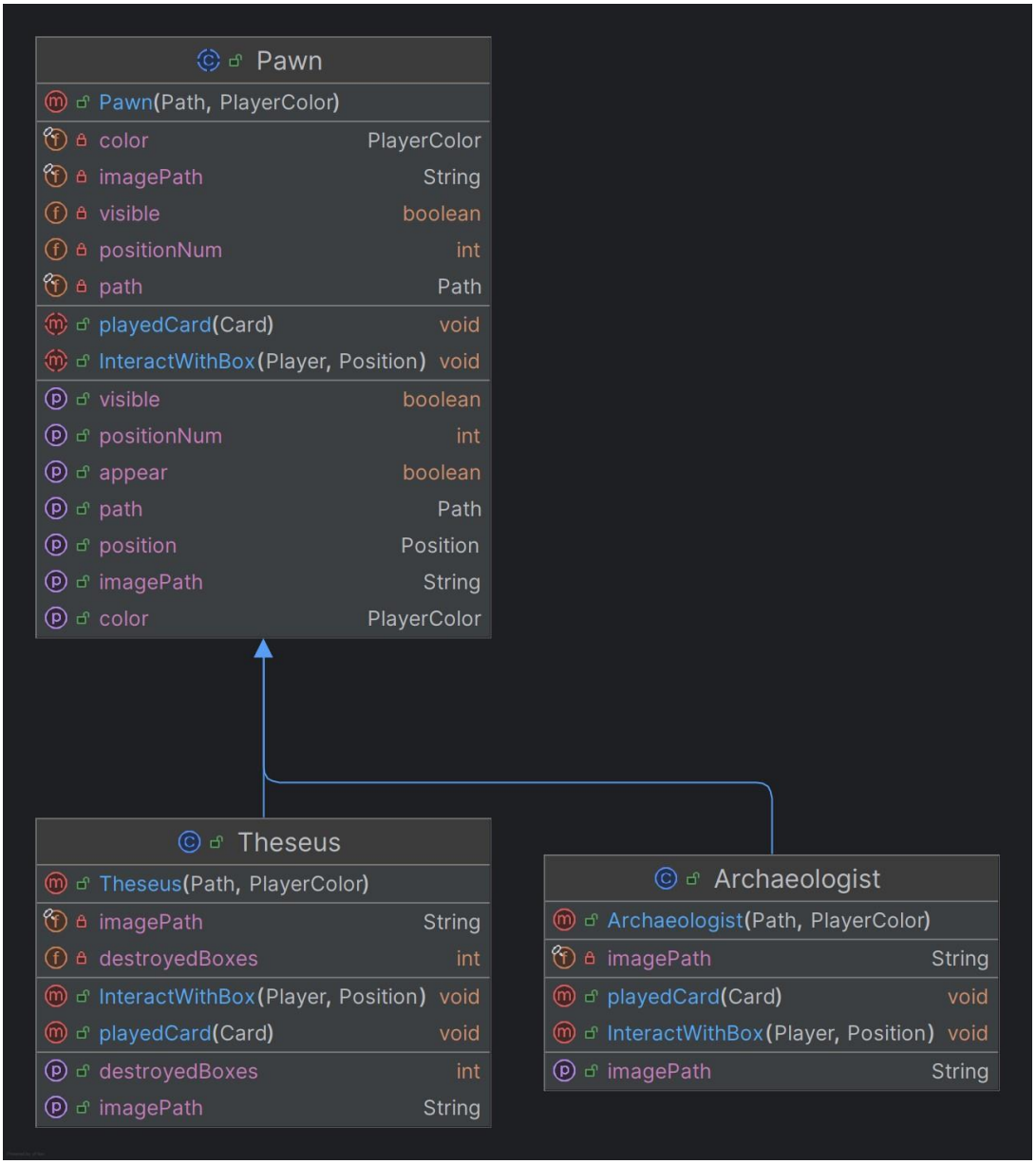
This is a subclass of the Pawn class. It represents the Archaeologist. It extends the Pawn and uses the super command to gain access to the pawn class and initializes its color and path.

It overrides the abstract methods of the Pawn class : playedCard and InteractWithBox, also overrides the getImagePath(). Doesn't have any other methods.

The attributes:

1. private final String imagePath (the image of the archaeologist)

This is a representation of the pawn package in UML diagram:



Exceptions package

Class IllegalMoveException

This is an exception when an illegal move is made during run time.

Player package

Class PlayerColor

It represents an enum class that contains the colors of the player (RED, GREEN)

Class Player

This class represents the Player in the game. Each player have a color, name, score, a set of hands (8 cards), a set of rare finds and murals that it captures during the game, count for how many statuettes (simple finds) it captured, a list of pawns (3 Archaeologists 1 Theseus), and a different sound that it playes during its turn.

When a player is created it will have a score 0 and no findings and 8 cards in its hand.

The Attributes:

1. private final PlayerColor color; // the color of the player
2. private final String name; // the name of the player
3. private int score; // the score of the player
4. private boolean turn; // to see if it's the player's turn
5. private List<Card> hand; // the hand of the player (its cards)
6. private List<RareFinds> rareFinds; // the rare finds that the player captured
7. private List<Murals> murals; // the murals that the player captured
8. private int statuettesCount; // the number of statuettes that the player captured
9. private List<Pawn> pawns; // the pawns of the player
10. private final String soundPath; // the sound of the player
11. private int availableArchaeologists = 3; // index of how many archaeologists left that the player can create
12. private boolean theseusAvailable = true; // index to check whether we chose a Theseus or not yet

The Methods:

1. public String getName() //Accessor Returns the name of the player
2. public PlayerColor getColor() //Accessor Returns the color of the player
3. public int getScore() //Accessor Returns the score of the player
4. public void setScore(int score) //Transformer Sets (updates) the score of the player
5. public boolean isTurn() //Observer
   Returns True if it is the player's turn, false otherwise.
6. public void setTurn(boolean turn) //Transformer
   Sets the turn of the player
7. public List<Card> getHand() //Accessor

Return the hand (the cards) of the player

8. public void drawCard(CardDeck cardDeck) method that draws a card from the card deck and adds it to the player's hand
9. public void discardCard(Card card, DiscardDeck discardDeck) method to discard a card from the player's hand
10. public Card playCard(int cardIndex) play a card from the player's hand
11. public List<RareFinds> getRareFinds() //Accessor Returns the rare finds of the player
12. public List<Murals> getMurals() //Accessor Returns the murals of the player
13. public void captureFind(Findings finding) capture a finding and adds it to its corresponding list
14. public int getStatuettesCount() //Accessor
    Returns the number of statuettes that the player captured
15. public void captureStatuette() // Transfomer Increments the number of Statuettes
16. public void createArch(Path path)
    Creates an archaeologist and adds it to the pawn list
17. public void createTheseus(Path path)
    Creates a Theseus and adds it to the pawn list
18. public int getAvailableArchaeologists() //Accessor
    Returns the number of available archaeologist that we can create
19. public boolean isTheseusAvailable() //Observer
    Returns true if we haven't created a theseus yet, false otherwise.
20. public void resetPlayer() resets the player's data
21. public List<Archaeologist> getArchaeologists() //Accessor
    Returns the Archaeologists in the pawn's list
22. public Theseus getTheseus() //Accessor Returns the Theseus in the pawn's list
23. public List<Pawn> getPawns() //Accessor Returns all the pawns in the pawn list

3. Η Σχεδίαση και οι Κλάσεις του Πακέτου Controller

This Class essentially is the mind of the game. We can also call it the Game class. Its responsible for creating a new game, 2 players, the board, the card deck, discard deck, and all the findings of the game. Also it is the connection between the view and the model. Essentially it takes the Players choices through the graphics (for example pushing the card deck button) and perform the action needed to play by the rules of the game.

When a controller is created, it creates 2 players with 0 score 8 cards each , a card deck that is shuffled, an empty discard deck, a board that has the 4 paths which each has 9 positions, and it initializes all the findings of the game and puts them in the corresponding list.

Every moment of the game it keeps calculating for any winning conditions and when they are met, the game will end and will calculate who is the winner.

The attributes:

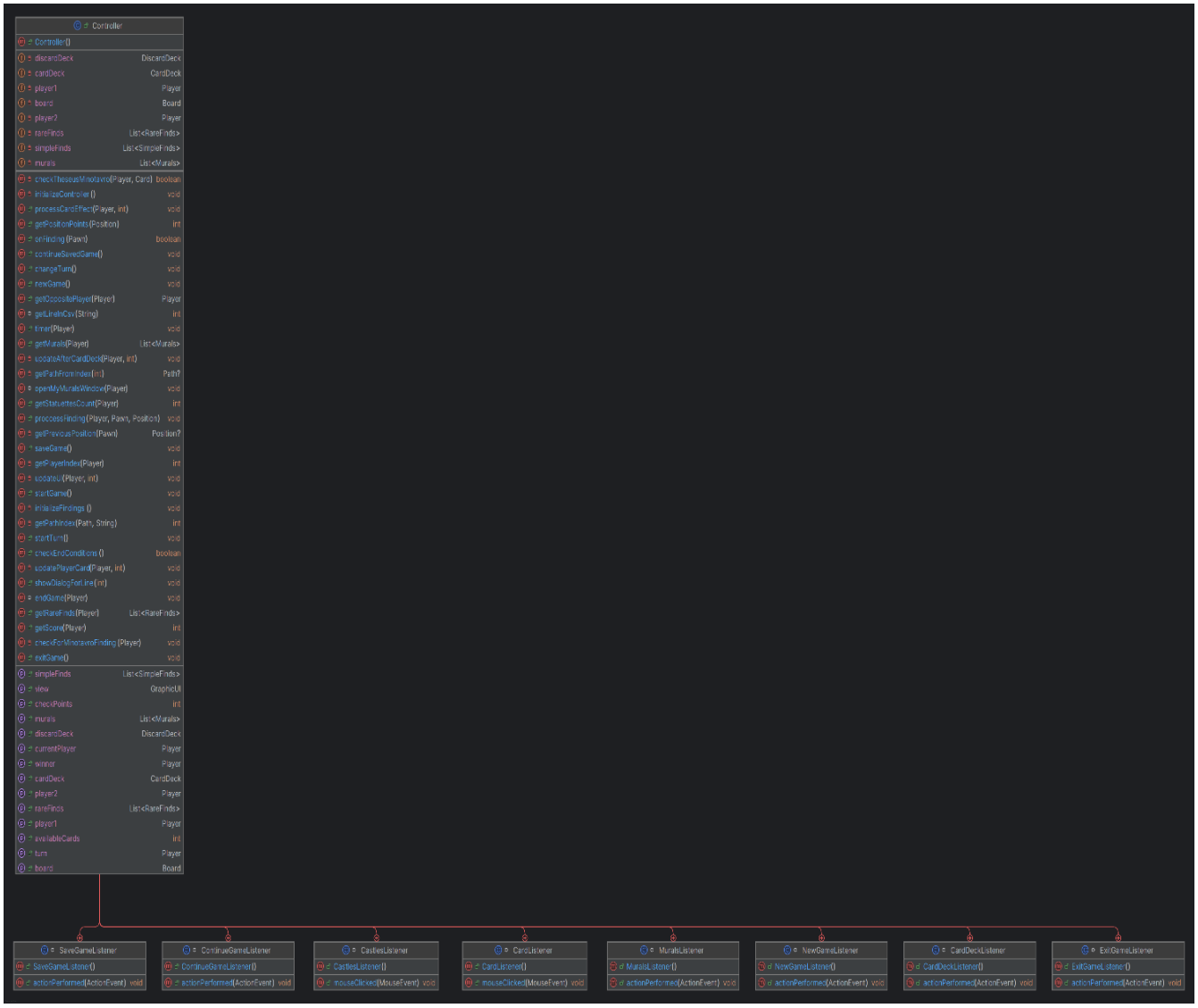
1. private Player player1; // first player of the game
2. private Player player2; // second player of the game
3. private Board board; // the board of the game
4. private final CardDeck cardDeck; // the card deck of the game
5. private DiscardDeck discardDeck; // the discard deck of the game
6. private List<RareFinds> rareFinds; // the rare finds of the game (4)
7. private List<SimpleFinds> simpleFinds; // the simple finds of the game
8. private List<Murals> murals; // the murals of the game (6)

The methods:

1. private void initializeFindings() initializes all the findings of the game and putting them in the corresponding list
2. public void newGame() Creates a new game
3. public void saveGame() saves the game
4. public Controller continueSavedGame() Continue a saved game
5. public void exitGame() exit the game
6. public void startGame() start of the game
7. public Player currentPlayer() //Accessor Returns the current player that has the turn
8. public Player startTurn()
   Randomly pick a player to have the turn at the start
9. public void endTurn(Player player) End the turn of a player
10. public Player seeTurn() // observer Returns the player that has the turn
11. public void changeTurn() changes the turn
12. public boolean checkEndConditions()
    Returns true if any winning condition is met, false otherwise

13. public void processCardEffect(Card card, Player player, Path path) a player playes a card in a specific path
14. public void movePawn(Pawn pawn, Path path) moves the pawn that is effected by the card
15. public boolean onFinding(Pawn pawn)
    Returns true if a pawn is on a finding
16. public void timer(Player player) the timer of the player's turn 17. public Player winner()
    Returns the player who won
18. public Player getPlayer1() //Accessor
    Returns the first player
19. public Player getPlayer2() //Accessor Returns the second player
20. public Player getOpponentPlayer(Player player) //Accessor Returns the opponent
21. public void setScore(Player player, int score) // Transformer Sets (updates) the score of the player
22. public int getScore(Player player) //Accessor returns the score of the player
23. public int getStatuettesCount(Player player) //Accessor returns the statuettes count of the player
24. public List<RareFinds> getRareFinds(Player player) //Accessor (not needed) Returns the rare finds of the player
25. public List<Murals> getMurals(Player player //Accessor (not needed) Returns the murals of the player
26. public int getPositionPoints(Position position) //Accessor (I don't think it is needed because we have a getPoints accessor in the position class) Returns the points of the position
27. public Board getBoard() //Accessor Returns the board
28. public CardDeck getCardDeck() //Accessor Returns the card deck
29. public int getAvailableCards() //Accessor
    Returns the number of available cards left in the game
30. public DiscardDeck getDiscardDeck() //Accessor Returns the discard deck
31. public List<RareFinds> getRareFinds() //Accessor
    Returns the Rare finds of the game
32. public List<SimpleFinds> getSimpleFinds() //Accessor Returns the Simple Finds of the game
33. public List<Murals> getMurals() //Accessor Returns the Murals of the game

Here is a UML of the controller

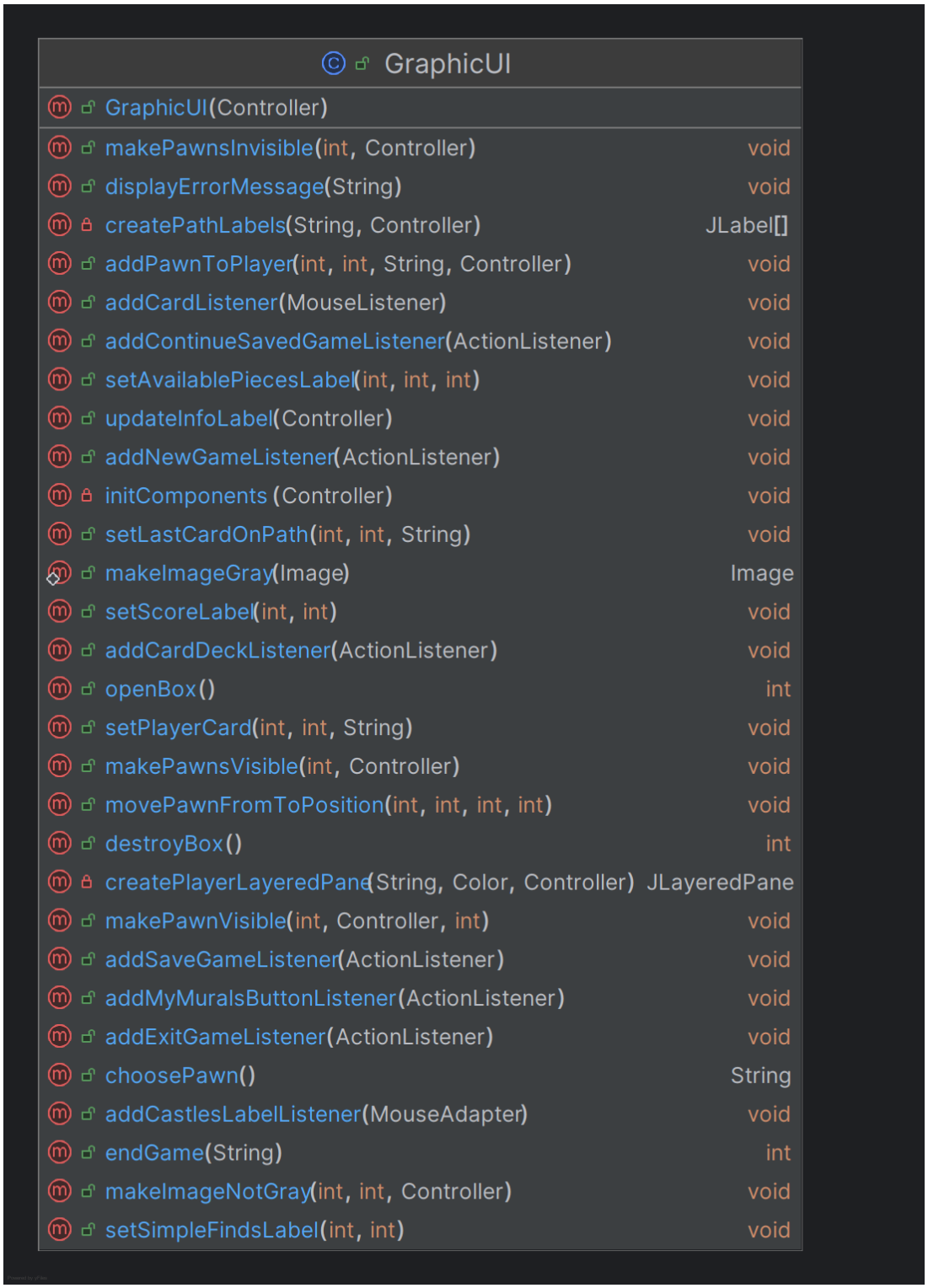

Controller
Controller()
discardDeck DiscardDeck
cardDeck CardDeck
player1 Player
board Board
player2 Player
rareFinds List<RareFinds>
simpleFinds List<SimpleFinds>
murals List<Murals>
checkTheseusMinotavro(Player, Card) boolean
initializeController() void
processCardEffect(Player, int) void
getPositionPoints(Position) int
onFinding(Pawn) boolean
continueSavedGame() void
changeTurn() void
newGame() void
getOppositePlayer(Player) Player
getLineInCsv(String) int
timer(Player) void
getMurals(Player) List<Murals>
updateAfterCardDeck(Player, int) void
getPathFromIndex(int) Path?
openMyMuralsWindow(Player) void
getStatuettesCount(Player) int
proccessFinding(Player, Pawn, Position) void
getPreviousPosition(Pawn) Position?
saveGame() void
getPlayerIndex(Player) int
updateU(Player, int) void
startGame() void
initializeFindings() void
getPathIndex(Path, String) int
start(Turn) void
checkEndConditions() boolean
updatePlayerCard(Player, int) void
showDialogForLine(int) void
endGame(Player) void
getRareFinds(Player) List<RareFinds>
getScore(Player) int
checkForMinotavroFinding(Player) void
exitGame() void
simpleFinds List<SimpleFinds>
view GraphicUI
checkPoints int
murals List<Murals>
discardDeck DiscardDeck
currentPlayer Player
winner Player
cardDeck CardDeck
player2 Player
rareFinds List<RareFinds>
player1 Player
availableCards int
turn Player
board Board
SaveGameListener
SaveGameListener()
actionPerformed(ActionEvent) void
ContinueGameListener
ContinueGameListener()
actionPerformed(ActionEvent) void
CastlesListener
CastlesListener()
mouseClicked(MouseEvent) void
CardListener
CardListener()
mouseClicked(MouseEvent) void
MuralsListener
MuralsListener()
actionPerformed(ActionEvent) void
NewGameListener
NewGameListener()
actionPerformed(ActionEvent) void
CardDeckListener
CardDeckListener()
actionPerformed(ActionEvent) void
ExitGameListener
ExitGameListener()
actionPerformed(ActionEvent) void

4. Η Σχεδίαση και οι Κλάσεις του Πακέτου View

The view package is responsible for implementing the graphical user interface GUI of the game. It provides the visual representation of the game and allow the 2 players to interact with its components. It has one class: GraphicUI Class, which creates the game window and initializes its components.
It has a JMenuBar which is the menu of the game (on top). It has a JLayeredPane for the game Board which will be in the central and has all the paths and the pawns on the positions. The paths are JLabel[]. Also the central part will have the Card deck button which will allow the players to draw a card, and the discard deck button which allow the players to discard a card. It has a JLayeredPane for the 2 player areas, each one display its cards as JButton, and includes labels for information of the score, rare finds, the number of statuettes. Also this class has a JOptionPane to display a message during gameplay.

It keeps listening for an interaction such as CardListener for when we press on a card we want to play (using left click). And to discard a card using right click on the card. DrawCardListener for when we press on the card deck to draw a card, MuralsListener when we want to view the murals it opens a window. MenuListener for when we want to press any button on the menu. When any Action Listener is executed it sends to the controller class to handle all the necessary changes and it should update the screen.

This is the UML diagram of the view package. Which has all the methods I used in the GraphicsUI class:


GraphicUI
GraphicUI(Controller)
makePawnsInvisible(int, Controller) void
displayErrorMessage(String) void
createPathLabels(String, Controller) JLabel[]
addPawnToPlayer(int, int, String, Controller) void
addCardListener(MouseListener) void
addContinueSavedGameListener(ActionListener) void
setAvailablePiecesLabel(int, int, int) void
updateInfoLabel(Controller) void
addNewGameListener(ActionListener) void
initComponents (Controller) void
setLastCardOnPath(int, int, String) void
makeImageGray(Image) Image
setScoreLabel(int, int) void
addCardDeckListener(ActionListener) void
openBox() int
setPlayerCard(int, int, String) void
makePawnsVisible(int, Controller) void
movePawnFromToPosition(int, int, int, int) void
destroyBox() int
createPlayerLayeredPane(String, Color, Controller) JLayeredPane
makePawnVisible(int, Controller, int) void
addSaveGameListener(ActionListener) void
addMyMuralsButtonListener(ActionListener) void
addExitGameListener(ActionListener) void
choosePawn() String
addCastlesLabelListener(MouseAdapter) void
endGame(String) int
makeImageNotGray(int, int, Controller) void
setSimpleFindsLabel(int, int) void

5. Λειτουργικότητα (Β Φάση)

In phase B I implemented each method. There were some methods that I put in phase A that I didn't use. And also there were some that I forgot to put that I realized I needed. Added the playersMusic class in the player package. That have the paths of the players music and have a method playMusicForPlayer that stops the music that was already playing and start a new Music in loop. Added the lastCard of each path in the player's class. Also added other methods in the controller and the GraphicsUI classes that helped me implement the game. But other than that the structure was quite similar to phase A.

The controller is the one in control. Every interaction with the Gui the controller handles it by calling methods from the model. By using inner classes like CardListener that extends MouseAdapter for having the ability to differentiate between a left click(plays a card) and right click(discards a card) . CardDeckListener that implements Actionlistener. NewGame listener, ExitGameListener, continueSavedGameListener, SavegameListener, MuralsListener, castlesListener for when a player press the last position of each path to get information about the path.

Also now each class in the model implements Serializable and that is in order to save the game and continue it.

6. Συμπεράσματα

One thing I need to mention is that I put an image path for the card, pawn … etc. Instead of that I could've done a method for example toString() and deal with the image paths in the view package. But I didn't have time to change everything so I didn't do it.

7. README:

PhaseB_5299

Overview:

PhaseB_5299 is the second phase of implementing the game "Searching for the Lost Minoan Palaces". It is a Java-based game project designed using the Model-View-Controller (MVC) architecture.

The game starts by picking a player randomly to have the first turn. Each player have 8 cards. and there is 4 paths in the board of the game. Each player have 4 pawns (3 archaeologists and 1 Theseus) and each one will have its own path. To start a path you should throw a Simple Card. the simple card moves the pawns 1 step. The Ariadni card 2 moves them 2 steps. and the Minotaur Card move the opponents Archaeologist 2 steps behind and make the Theseus not move for an entire round. The player have to either play a card or discard a card then he has to draw a card from the card deck. That's how each round will work. Also, there's positions that have findings, each finding have different points. And also each position that the pawn is on have different points. if an Archaeologist is on a position it counts the same as its points. but if a Theseus is on a position it counts the double. The game ends if 4 pawns reach the checkpoint or the card deck has finished. The player with the most points wins.

Some instructions to play the game:

Right-click on a card to discard it, left-click on a card to play it, and click on the label of the last position of any of the four paths (the four castles) to get information about the path. Also, if a player acquire a finding it will show info about it (only the Archaeologists because Theseus destroys them). Also, if a player tries to play an illegal move, he won't be able to, and there will be an error message.

Setup Instructions:

To get started, ensure JDK 8 or higher is installed on your system. Install Apache Ant. Save all source files in UTF-8 encoding to ensure Greek characters display correctly.

We use Apache ant to build and run the project.

To clean the project run: "ant clean"

To Compile the project run: "ant compile"

To run the project without a jar file run: "ant run"

To package the project into a jar file run: "ant jar"

To run the jar file run: "ant run-jar"

Enjoy playing!

ΤΕΛΟΣ