Πανεπιστήμιο Κρήτης –Τμήμα Επιστήμης Υπολογιστών

ΗΥ 252– Αντικειμενοστραφής Προγραμματισμός

Διδάσκων: Ι. Τζίτζικας

Χειμερινό Εξάμηνο 2020-2021

STRATEGO PHASE B

Μιχάλης Ιεροδιακόνου
csd4773
14/01/2023

Περιεχόμενα:

1. Εισαγωγή

Σε αυτή την υλοποίηση το παιχνιδιου “Stratego Ice Vs Fire” θα βασιστούμε στο MVC (Model View Controller) μοντέλο. Η διεπαφή με τον χρήστη θα υλοποιείται στο View το οποίο μέσω του Controller θα συνδέεται στο Model του παιχνιδιού. Το παιχνίδι ξεκινά με την εφαρμογή Game.java η οποία δημιουργά ένα αντικείμενο τύπου Controller μέσα από τον οποίο ελέγχονται όλες οι διεργασίες του παιχνιδιού. Η φάση Β χρειάστηκε αρκετή δουλειά και το τελικό αποτέλεσμα με έχει αφήσει αρκετά ευχαριστημένο.

2. Packages

1. Model

Εδώ περιέχονται όλες οι απαραίτητες κλάσεις για την δημιουργία, υλοποίηση και ενημέρωση του παιχνιδιου.

SubPackage Coordinates:

1. Class Coordinates:

Εχω υλοποιήσει αυτή την κλάση για διευκόλυνση στη μετακίνηση πληροφοριών μεταξύ μεθόδων. Αποτελεί απλοποίηση μεταφοράς x,y ακεραίων αριθμών που αντιπροσωπεύουν συντεταγμένες.

Πεδία:

- `private int y;`
- `private int x;`

Μεθόδοι:

1. `public void setX(int x) {this.y = x;}`
   Transformer:
   Set X-coordinate
2. `public void setY(int y) {this.y = y;}`
   Transformer:
   Set Y-coordinate
3. `public int getX() {return x;}`
   Accessor:
   Get X-coordinate

4. `public int getY() {return y;}`
   Accessor:
   Get Y-coordinate
5. `public boolean isValid() {...}`
   Accessor:
   Returns true if the coordinate is inside the board's (8x10) borders
6. `public String toString() { return "(" + x + "," + y + ")";}`
   Accessor:
   Returns x and y values as (x,y) representation

SubPackage Pieces:

1. Abstract Class Piece:

Αφαιρετική Κλάση για την υλοποίηση των πιονιών του παιχνιδιού. Κάθε πιόνι έχει συντεταγμένη (x,y), την κατάταξη του, την τοποθεσία της εικόνας που του αντιστοιχεί, σε ποιά ομάδα ανήκει ,αν είναι ζωντανό ή όχι, αν είναι η σειρά του (flipped or not) το path της εικόνας, 2 αντικείμενα ImageIcon για την εικόνα του πιονιού οταν φαίνεται και για την σειρά του αντίπαλου παίκτη, ανάλογα με την ομάδα του. Τα δύο final int στο τέλος είναι το μέγεθος που πρέπει να γίνουν scaled οι εικόνες.

Πεδία:

- `private int rank;`
- `private String imagePath;`
- `private boolean hasRevived;`
- `private boolean isBlue;`
- `private boolean isDead;`
- `private Coordinates coordinates;`
- `private boolean flipped = false;`
- `private ImageIcon pieceImage;`
- `private ImageIcon hiddenImage;`
- `private static final int buttonWidth = 105;`
- `private static final int buttonHeight = 95;`

Μεθόδοι:

1. `public void setX(int x) {this.y = x;}`
   Transformer:
   Set X-coordinate
2. `public void setY(int y) {this.y = y;}`
   Transformer:
   Set Y-coordinate
3. `public void setCoordinates(Coordinates coordinates) {this.coordinates = coordinates;}`

Transformer:
Set coordinates

4. `public void setImagePath(String imagePath) {this.imagePath = imagePath;}`
   Transformer:
   Set Image Path as sent from controller
5. `public void setDead() {isDead = true;}`
   Transformer:
   Set Piece to dead
6. `public void setFlipped(boolean flipped) {isFlipped = flipped;}`
   Transformer:
   Set Piece isFlipped to the given boolean
7. `public void setHiddenImage(String path)throws PathNotFoundException {...}`
   Transformer:
   Generates a scaled instance of the Piece's hidden image
8. `public void setPieceImage()throws PathNotFoundException {...}`
   Transformer:
   Generates a scaled instance of the Piece's image
9. `public void isRevived() {isDead = true;}`
   Transformer:
   Set Piece isDead to true as it has been rescued
10. `public void revivedSomeboody() {hasRevived = true;}`
    Transformer:
    Set Piece hasRevived to true as this piece rescued another piece
11. `public int getX(){return coordinates.getX();}`
    Accessor:
    Get X-Coordinate of the piece
12. `public int getY(){return coordinates.getY();}`
    Accessor:
    Get Y-Coordinate of the piece
13. `public Coordinates getCoordinates() {return coordinates;}`
    Accessor:
    Get coordinates of the piece
14. `public int getRank() {return rank;}`
    Accessor:
    Returns the Rank of the piece (0-11, 0 is Flag, 11 is Trap)
15. `public String getImagePath() {return imagePath;}`
    Accessor:
    Returns the image Path so it can render it as an ImageIcon on the board
16. `public void getHiddenImage(String path){...}`
    Accessor:
    Returns the instance of the Piece's hidden image
17. `public void getPieceImage(String path){...}`
    Accessor:
    Returns the instance of the Piece's image
18. `public boolean isDead() {return isDead;}`
    Accessor:
    Returns true if this Piece is Dead
19. `public boolean isFlipped() {return isFlipped;}`

Accessor:
Returns true if this Piece is Flipped

20. `public boolean hasRevived() {return hasRevived;}`
Accessor:
Returns true if this Piece has already rescued another piece or not
21. `public String toString() {return "Rank: "+this.rank;}`
Accessor:
Returns the rank of the piece in string form
22. `public abstract List<Coordinates> getPossibleMoves(Board board,int mode);`
Accessor:
Returns an Array List of Coordinates which the piece can move to based on the board it is placed in

2. Class ImmovablePiece: (extends Piece)

Κλάση για την υλοποίηση μη κινούμενων πιονιών του παιχνιδιού. Σε αυτή την κλάση υπάγεται η Σημαία και η Παγίδα.

Πεδία:

- `private boolean isFlag;`

Μεθόδοι:

1. `public boolean isFlag() {return isFlag;}`
Accessor
Returns true if this ImmovablePiece isFlag
2. `public abstract List<Coordinates> getPossibleMoves(Board board,int mode);`
Accessor:
Returns null as immovable pieces cannot move hence don't have possible moves

3. Class MovablePiece: (extends Piece)

Κλάση για την υλοποίηση μετακινούμενων πιονιών του παιχνιδιού. Αυτή η κλάση επεκτείνει την παραπάνω κλάση Piece προσθέτωντας 3 νέες μεθόδους μοναδικές για τα κινούμενα πιόνια.

Μεθόδοι:

1. `public List<Coordinates> getPossibleMoves(Board board,int mode) {return new ArrayList<Coordinates>();}`

Accessor

Calculates and Returns a list of coordinates (max 4) in which this piece can move to, also depending on the mode (i.e. no retreat)

2. `public void move(Coordinates newPos) throws InvalidCoordinatesException {}`

   Transformer

   Set MovablePiece's new Coordinates from one of the possibleMoves calculated below and throw exception if they are out of bounds

3. `public void attack(Piece Enemy) throws DeadPieceException{}`

   Transformer

   Simulates the attack between this MovablePiece and an Enemy Piece and sets Dead one or both depending on their respective Rank

4. Class SpecialMovablePiece: (extends MoveablePiece)

Κλάση για την υλοποίηση Ειδικών μετακινούμενων πιονιών του παιχνιδιού. Αυτή η κλάση επεκτείνει την παραπάνω κλάση Piece και η χρήση της είναι για να για το override των μεθόδων getPossibleMoves ή attack της MovablePiece.

5. Class Slayer: (extends SpecialMovablePiece)

Κλάση για την υλοποίηση του Slayer ο οποιός υλοποιεί μια διαφορετική μέθοδο attack.

Μεθόδοι:

`@Override`

1. `public void attack(Piece Enemy) throws DeadPieceException{}`

   Transformer:

   Simulates the attack between this MovablePiece and an Enemy Piece and sets Dead one or both depending on their respective Rank

6. Class Scout: (extends SpecialMovablePiece)

Κλάση για την υλοποίηση του Ανιχνευτή/Scout ο οποιός υλοποιεί μια διαφορετική μέθοδο getPossibleMoves.

Μεθόδοι:

`@Override`

1. `public List<Coordinates> getPossibleMoves() {return new ArrayList<Coordinates>();}`

   Accessor:

   Calculates and Returns a list of coordinates (max 4) in which this piece can move to

<u>7. Class Dwarf:</u> (extends SpecialMovablePiece)

Κλάση για την υλοποίηση του Dwarf ο οποιός υλοποιεί μια διαφορετική μέθοδο attack.

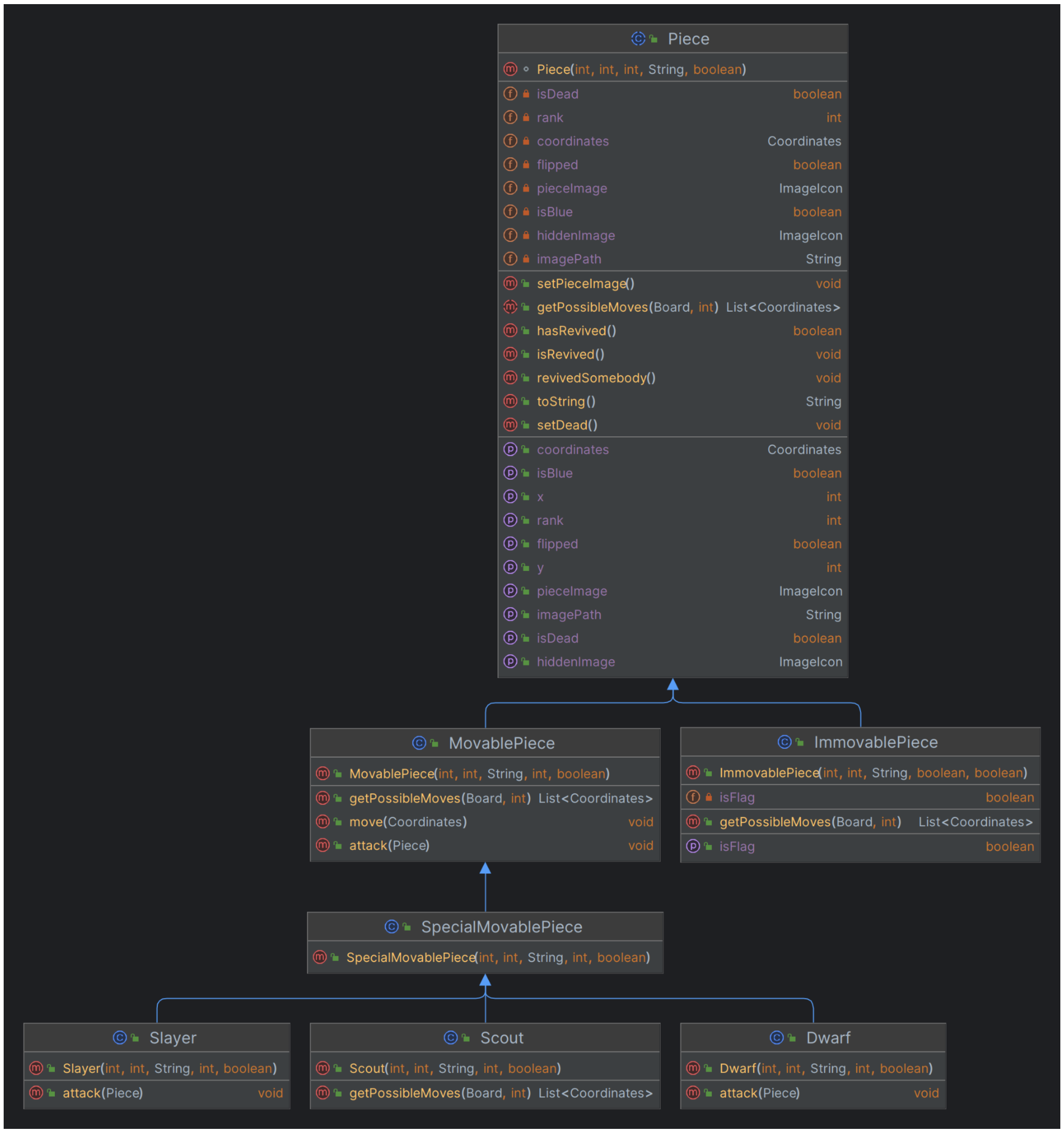
<u>Μεθόδοι:</u>

@Override

1. `public void attack(Piece Enemy) throws DeadPieceException{}`

   <u>Transformer:</u>

   Simulates the attack between this MovablePiece and an Enemy Piece and sets Dead one or both depending on their respective Rank



SubPackage Spot:

1. Class Spot: (extends Coordinates)

Κλάση για την υλοποίηση των “Θεσεων” του ταμπλό(board) του παιχνιδιού. Αυτή η κλάση επεκτείνει την παραπάνω κλάση Συντεταγμένων άρα κάθε Θέση/Spot έχει συντεταγμένη (x,y), το πιόνι σε αυτή τη θέση και αν η θέση αυτή είναι λίμνη ή όχι.

Πεδία:

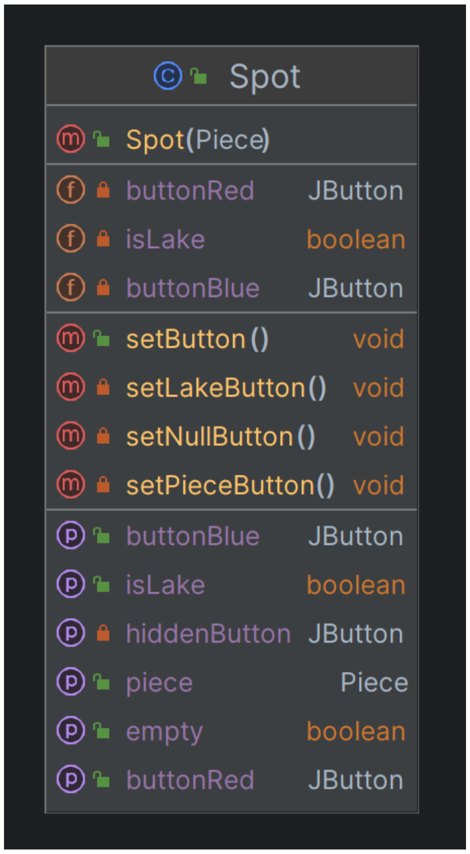
- `private Piece s_Piece;`
- `private boolean isLake;`
- `private final JButton buttonRed = new JButton();`
- `private final JButton buttonBlue = new JButton();`
- `private static final int buttonWidth = 105;`
- `private static final int buttonHeight = 95;`

Μεθόδοι:

1. `public void setPiece(Piece s_Piece) {this.s_Piece = s_Piece;}`
   Transformer:
   Set the Piece of this Spot to either an object Piece or null
2. `public void setLake(boolean isLake) {this.isLake = isLake;}`
   Transformer:
   Set true if Lake or false else
3. `public void setButton() {...}`
   Transformer: Sets up both buttons of the spot with these methods
   a. `private void setPieceButton() {...}`
   b. `private void setLakeButton() {...}`
   c. `private void setNullButton() {...}`
   d. `private void setHiddenButton() {...}`
4. `public Piece getPiece() {return s_Piece;}`
   Accessor:
   Returns the occupying this Spot at the moment
5. `public boolean isLake() {return isLake;}`
   Accessor:
   Returns if the Spot is a Lake(Inaccessible) or not(accessible)
6. `public boolean isEmpty() {}`
   Accessor:
   Returns if the Spot has a piece in it or not
7. `public boolean getButtonRed() {return buttonRed;}`
   Accessor:
   Returns the button the red player is seeing
8. `public boolean getButtonBlue() {return buttonBlue;}`

Accessor:

a. Returns the button the blueplayer is seeing



Spot	
Spot(Piece)	
buttonRed	JButton
isLake	boolean
buttonBlue	JButton
setButton ()	void
setLakeButton ()	void
setNullButton ()	void
setPieceButton ()	void
buttonBlue	JButton
isLake	boolean
hiddenButton	JButton
piece	Piece
empty	boolean
buttonRed	JButton

SubPackage Player:

1. Class Player:
Κλάση για την υλοποίηση των παικτών του παιχνιδιού. Είτε Κοκκινος () είτε Μπλε (). Κρατάνε πληροφορίες για τα πιόνια τους, ποιά είναι ζωντανά ποιά όχι, ποιά πιόνια του αντιπάλου έχουν καταλάβει

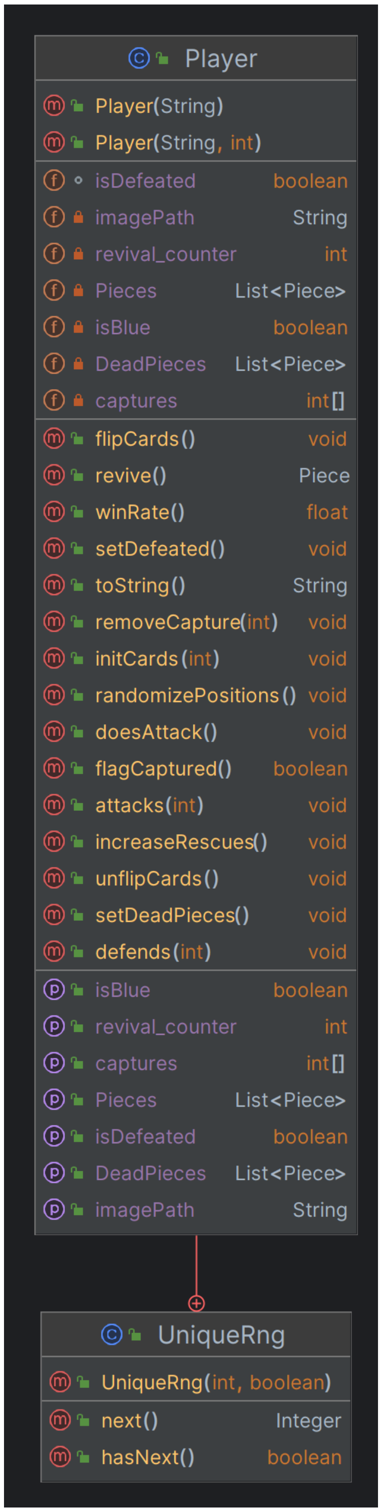
Πεδία:
- `private String m_name;`
- `private boolean isBlue;`
- `private int revival_counter;`
- `private String imagePath;`
- `private int m_mode;`
- `private String m_HiddenImagePath;`
- `private List<Piece> Pieces;`
- `private List<Piece> DeadPieces;`
- `boolean isDefeated = false;`
- `private List<Piece> Pieces;`
- `private List<Piece> DeadPieces;`
- `private int attackCount;`
- `private int successfulAttacks;`
- `private int[] captures;`

Μεθόδοι:
1. `public void initCards(int mode) {}`
   Transformer:
   Pieces are initialized all with coordinates (0,0) to later be randomized by the board
2. `public void randomizePositions() {}`
   Transformer:
   Pieces Coordinates are randomized depending if the Player is Blue or Red, top and bottom of the board respectively
3. `public void setDefeated() {}`
   Transformer:
   Sets the player's field isDefeated to true
4. `public void makeMove(Coordinates newPos) {}`
   Transformer:
   Changes the position or state of one of its Pieces
5. `public void attacks() {}`
   Transformer:
   Updates the attacking stat fields and increases the captured piece of rank (=index) when the attack was successful
6. `public void defends() {}`
   Transformer:
   Increases the captured piece of rank (=index) when the defense was successful
7. `public void doesAttack() {}`

Transformer:
Just updates the stats on an unsuccessful attack
8. `public void flipCards() {}`
Transformer:
Sets each each piece the of the player as flipped
9. `public void unflipCards() {}`
Transformer:
Sets each each piece the of the player as not flipped
10. `public void increaseRescues() {}`
Transformer:
Increments rescue counter
11. `public void removeCaptures(int index) {}`
Transformer:
Remove one of the captured enemy pieces with rank = index
12. `public float winRate() {}`
Accessor:
Calculates and returns the percentage of successful attacks made by this player
13. `public String getImagePath() {return ImagePath;}`
Accessor:
Returns the path for the folder of images of this Player
14. `public void setDeadPieces() {}`
Accessor:
Iterates the list of pieces and adds the dead ones on the dead pieces list
15. `public List<Coordinates> getDeadPieces() {}`
Accessor:
Returns the dead pieces list
16. `public List<Coordinates> getPieces() {}`
Accessor:
Returns the pieces list
17. `public int[] getCaptures(){}`
Accessor
Returns an array of integers corresponding to the ranks of the enemy pieces this player captured
18. `public boolean isBlue() {}`
Accessor:
Returns if the player is the blue player or not
19. `public boolean isDefeated() {}`
Accessor:
Returns if the player is defeated or not
20. `public int getRevival_counter() {}`
Accessor:
Returns the number of rescues this player made in this game
21. `public Piece revive() {}`
Accessor:
Generates a UI letting the user select which of the fallen pieces ranks he would like to rescue and returns that piece's instance from the dead pieces list to the board to manage the rescue
22. `public String toString(){}`

Accessor
Returns the name of the player

23. public class UniqueRng implements Iterator<Integer>{...}
Inner Class
Responsible for generating a sequence of random integers between 0-29 and 50-79 depending on the player's side



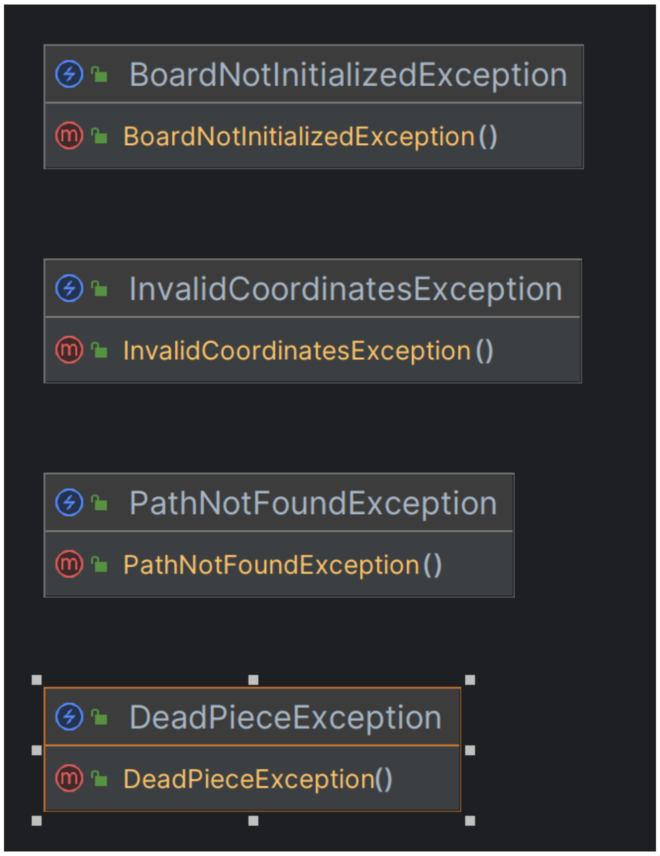
Player	
Player(String)	
Player(String, int)	
isDefeated	boolean
imagePath	String
revival_counter	int
Pieces	List<Piece>
isBlue	boolean
DeadPieces	List<Piece>
captures	int[]
flipCards()	void
revive()	Piece
winRate()	float
setDefeated()	void
toString()	String
removeCapture(int)	void
initCards(int)	void
randomizePositions()	void
doesAttack()	void
flagCaptured()	boolean
attacks(int)	void
increaseRescues()	void
unflipCards()	void
setDeadPieces()	void
defends(int)	void
isBlue	boolean
revival_counter	int
captures	int[]
Pieces	List<Piece>
isDefeated	boolean
DeadPieces	List<Piece>
imagePath	String

UniqueRng	
UniqueRng(int, boolean)	
next()	Integer
hasNext()	boolean

SubPackage Exceptions:

Κάποια Custom Exceptions για ειδικές περιπτώσεις.

1. Class BoardNotInitializedException:
   Σε περίπτωση που υπάρξει σφάλμα στην αρχικοποίηση του ταμπλό
2. DeadPieceException:
   Σε περίπτωση που υπάρξει σφάλμα στην αρχικοποίηση του ταμπλό
3. InvalidCoordinatesException:
   Σε περίπτωση που υπάρξει σφάλμα στην αρχικοποίηση του ταμπλό
4. PathNotFoundException:
   Σε περίπτωση που υπάρξει σφάλμα στην αρχικοποίηση του ταμπλό


BoardNotInitializedException
BoardNotInitializedException()
InvalidCoordinatesException
InvalidCoordinatesException()
PathNotFoundException
PathNotFoundException()
DeadPieceException
DeadPieceException()

SubPackage Board:

1. Class Board:

Κλάση για την υλοποίηση του ταμπλό του παιχνιδιού αποτελούμενο από ένα δισδιάστατο πίνακα Spot[ ][ ]. Σε αυτή την κλάση πραγματοποιείτε το μεγαλύτερο μέρος της αλληλεπίδρασης του χρήστη με το μοντέλο του παιχνιδιού καθώς εδώ βρίσκεται ο ActionListener τον οποίο ακολουθούν όλα τα JButtons ανεξαιρέτως παίκτη.

Πεδία:

- `private static Spot lastPressedPiece;`
- `private static Spot targetSpot;`
- `private Spot[][] spots;`
- `private int[] lastPressed;`
- `private Player playerBlue;`
- `private Player playerRed;`
- `private Piece pieceToRevive;`
- `private int m_mode;`
- `private int movables = 0;`
- `private boolean moveMade = false;`
- `private boolean attackMade = false;`
- `private boolean revivePending = false;`
- `private boolean reviveMade = false;`
- `private List<Coordinates> possibleCoordinates = new ArrayList<>();`
- `private List<JButton> possibleButtons = new ArrayList<JButton>();`

Μεθόδοι:

1. `public void initializeBoard() throws BoardNotInitialized {}`
   Transformer:
   Initialize the Spot array with all null pointers and lakes in the appropriate positions
2. `public void initButttonSpots() {}`
   Transformer:
   Initializes all the buttons by adding the action listener to both spots' buttons
3. `public void placePlayer(Player p){}`
   Transformer:
   Places the given player's (p) pieces on the board
4. `public void movePiece(Coordinates temp) throws InvalidCoordinatesException{}`
   Transformer:

Simulates movement and calls attack method when needed

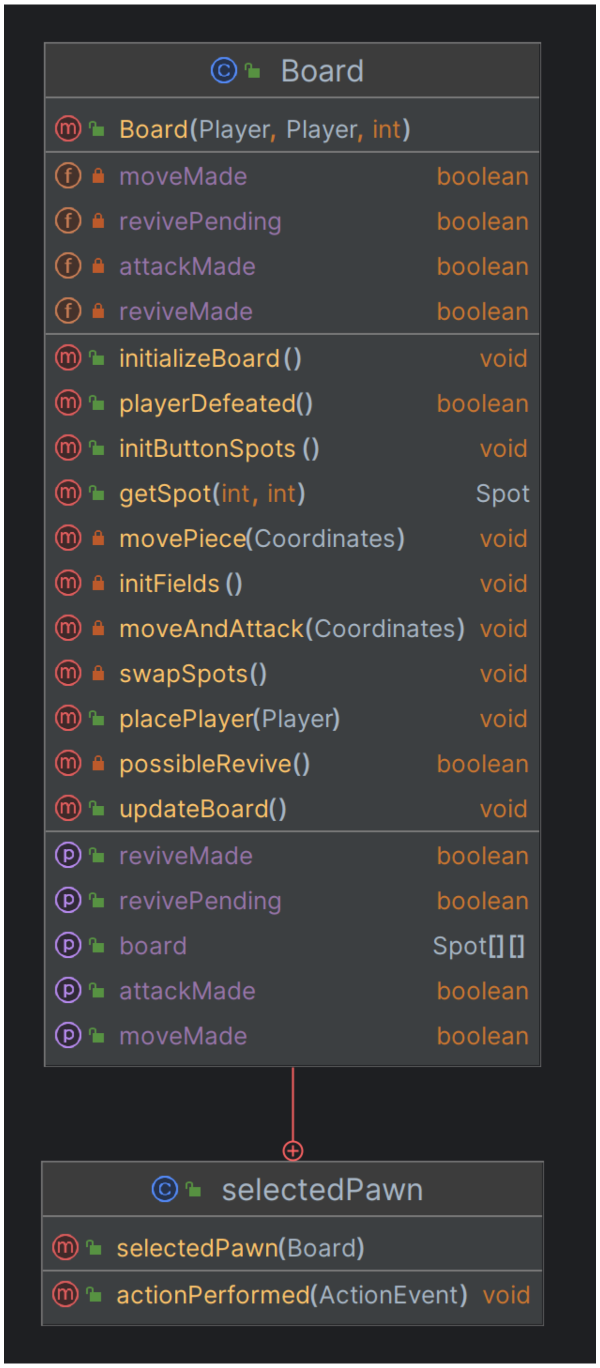
5. `public void swapSpots(){}`
   Transformer:
   Swaps the piece inhabiting the last pressed spot with the target spot
6. `public void initFields(){}`
   Transformer:
   Sets all the temporary board's fields to their default values
7. `public void moveAndAttack(Coordinates temp) throws InvalidCoordinatesException, DeadPieceException{}`
   Transformer:
   Simulates the attack and moves the piece attacking if it wins the battle
8. `public void updateBoard(Player blue, Player red ){}`
   Transformer:
   Updates the board based on both players' pieces
9. `public void setMoveMade(boolean moveMade ){}`
   Transformer:
   Sets the value of moveMade, the flag to show if a move was made in this turn, to the given value
10. `public void setAttackMade(boolean attackMade ){}`
    Transformer:
    Sets the value of attackMade, the flag to show if an attack was made in this turn, to the given value
11. `public void setRevivePending(boolean revivePending){}`
    Transformer:
    Sets the value of revivePending, the flag to show if a revive is pending this turn, to the given value
12. `public void setReviveMade(boolean reviveMade){}`
    Transformer:
    Sets the value of revive Made, the flag to show if the revive pending was made this turn, to the given value and resets the revive pending value to false
13. `public Spot getSpot(int x, int y) {return spots[x][y];}`
    Accessor:
    Returns the Spot in the given position
14. `public Spot[][] getBoard() {}`
    Accessor:
    Returns current state of the board
15. `private boolean possibleRevive() {...}`
    Accessor:
    Checks and returns true if the current player has a piece that can make a rescue happen or not
16. `public boolean getMoveMade( ){}`
    Accessor:
    Returns the value of moveMade, the flag to show if a move was made in this turn
17. `public boolean getAttackMade( ){}`
    Accessor:
    Returns the value of attackMade, the flag to show if an attack was made in this turn
18. `public boolean isRevivePending(){}`

Accessor:
Returns the value of revivePending, the flag to show if a revive is pending this turn

19. `public boolean isReviveMade(){}`
Accessor:
Returns the value of revive Made, the flag to show if the revive pending was made this turn

20. `public boolean playerDefeated(){}`
Accessor:
Calculates and returns if a player is defeated or not

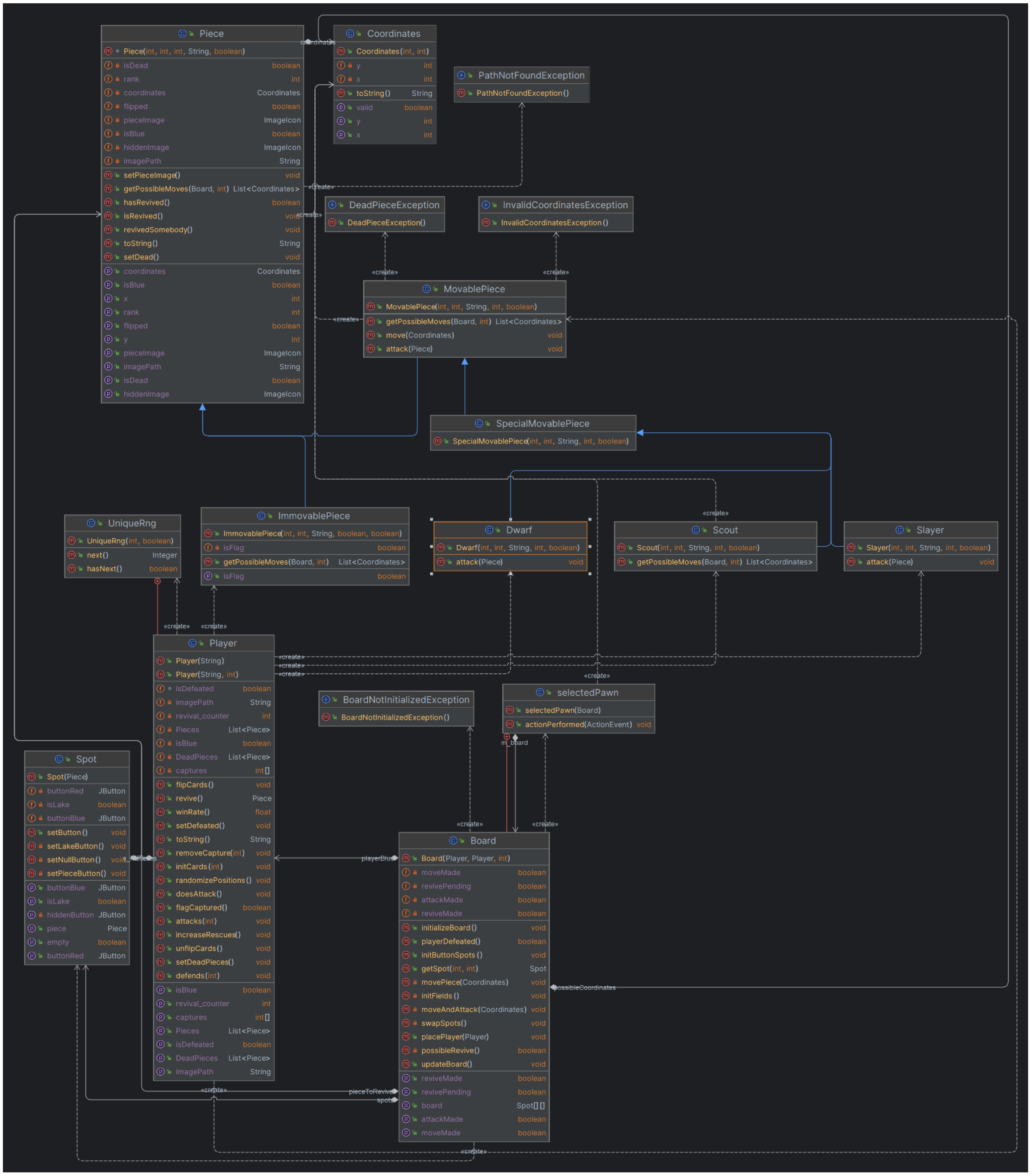
21. `public class selectedPawn implements ActionListener{...}`
Inner Class
Responsible for generating the action listener each button gets and is responsible for calculating and executing the largest part of the game which are movements and attacks



```
Board
Board(Player, Player, int)
moveMade                          boolean
revivePending                     boolean
attackMade                        boolean
reviveMade                        boolean
initializeBoard()                 void
playerDefeated()                  boolean
initButtonSpots()                 void
getSpot(int, int)                 Spot
movePiece(Coordinates)            void
initFields()                      void
moveAndAttack(Coordinates)        void
swapSpots()                       void
placePlayer(Player)               void
possibleRevive()                  boolean
updateBoard()                     void
reviveMade                        boolean
revivePending                     boolean
board                             Spot[][]
attackMade                        boolean
moveMade                          boolean

selectedPawn
selectedPawn(Board)
actionPerformed(ActionEvent)      void
```

<u>MODEL UML</u>


Piece
Piece(int, int, int, String, boolean)
isDead boolean
rank int
coordinates Coordinates
flipped boolean
pieceImage ImageIcon
isBlue boolean
hiddenImage ImageIcon
imagePath String
setPieceImage() void
getPossibleMoves(Board, int) List<Coordinates>
hasRevived() boolean
isRevived() void
revivedSomebody() void
toString() String
setDead() void
coordinates Coordinates
isBlue boolean
x int
rank int
flipped boolean
y int
pieceImage ImageIcon
imagePath String
isDead boolean
hiddenImage ImageIcon
Coordinates
Coordinates(int, int)
y int
x int
toString() String
valid boolean
y int
x int
PathNotFoundException
PathNotFoundException()
DeadPieceException
DeadPieceException()
InvalidCoordinatesException
InvalidCoordinatesException()
MovablePiece
MovablePiece(int, int, String, int, boolean)
getPossibleMoves(Board, int) List<Coordinates>
move(Coordinates) void
attack(Piece) void
SpecialMovablePiece
SpecialMovablePiece(int, int, String, int, boolean)
UniqueRng
UniqueRng(int, boolean)
next() Integer
hasNext() boolean
ImmovablePiece
ImmovablePiece(int, int, String, boolean, boolean)
isFlag boolean
getPossibleMoves(Board, int) List<Coordinates>
isFlag boolean
Dwarf
Dwarf(int, int, String, int, boolean)
attack(Piece) void
Scout
Scout(int, int, String, int, boolean)
getPossibleMoves(Board, int) List<Coordinates>
Slayer
Slayer(int, int, String, int, boolean)
attack(Piece) void
Player
Player(String)
Player(String, int)
isDefeated boolean
imagePath String
revival_counter int
Pieces List<Piece>
isBlue boolean
DeadPieces List<Piece>
captures int[]
flipCards() void
revive() Piece
winRate() float
setDefeated() void
toString() String
removeCapture(int) void
initCards(int) void
randomizePositions() void
doesAttack() void
flagCaptured() boolean
attacks(int) void
increaseRescues() void
unflipCards() void
setDeadPieces() void
defends(int) void
isBlue boolean
revival_counter int
captures int[]
Pieces List<Piece>
isDefeated boolean
DeadPieces List<Piece>
imagePath String
BoardNotInitializedException
BoardNotInitializedException()
selectedPawn
selectedPawn(Board)
actionPerformed(ActionEvent) void
Spot
Spot(Piece)
buttonRed JButton
isLake boolean
buttonBlue JButton
setButton() void
setLakeButton() void
setNullButton() void
setPieceButton() void
buttonBlue JButton
isLake boolean
hiddenButton JButton
piece Piece
empty boolean
buttonRed JButton
Board
Board(Player, Player, int)
moveMade boolean
revivePending boolean
attackMade boolean
reviveMade boolean
initializeBoard() void
playerDefeated() boolean
initButtonSpots() void
getSpot(int, int) Spot
movePiece(Coordinates) void
initFields() void
moveAndAttack(Coordinates) void
swapSpots() void
placePlayer(Player) void
possibleRevive() boolean
updateBoard() void
reviveMade boolean
revivePending boolean
board Spot[][]
attackMade boolean
moveMade boolean
coordinates
«create»
m_board
playerBlue
pieceToRevive
spot
possibleCoordinates

2. Controller

1. Class Controller:

Αυτη η κλάση θα αποτελεί τον τρόπο επικοινωνίας των 2 παικτών με το ταμπλό το οποίο θα στέλνει έπειτα για τη δημιουργία των γραφικών. Αποτελείται από κατά κύριο λόγω private classes με σκοπό ο χρήστης να έχει πρόσβαση σε όσο λιγότερο το δυνατό στοιχεία του παιχνιδιού.

Πεδία:

- `private int round = 0;`
- `private static boolean turnRed;`
- `private Player playerBlue;`
- `private Player playerRed;`
- `private int mode;`
- `private View view;`
- `private Board board;`
- `private LoadingScreen loading;`
- `private ModSelectionWindow msw;`
- `private winningFrame blueWins;`
- `private winningFrame redWins;`
- `private reviveSelectionFrame rsf=null;`

Μεθόδοι:

1. `public void startGame(){}`
   Transformer:
   Starts the game and goes through each phase
2. `private void init(){}`
   Transformer:
   Initializes the mod selection window and loading screen
3. `private void menu(){}`
   Transformer:
   Method turning on the GUI and waiting for the user to input the mods on the mod selection window
4. `private void createBoard(){}`
   Transformer:
   Creates the Players and adds them to the board it creates. Also now winning Frames are being initialized
5. `private void afterMenu(){}`
   Transformer:
   Method to visualize the transition between the selection screen and the game
6. `private void GameLoop(){}`
   Method which continuously checks if a player on the board is defeated and coordinates the attacks/moves on the board

7. `private void endGame(){}`
   Method for ending the game and showing the winning player screen
8. `private void updateLists(){}`
   Transformer:
   Used to update the dead pieces of each player after the attacks or moves
9. `private void nextTurn(){this.turnBlue = !this.turnBlue;}`
   Transformer:
   Swaps between the turn of blue and red player
10. `private void nextRound(){round++;}`
    Transformer:
    Increases round counter to go to the next round
11. `private class Thread_extended_class extends Thread{...}`
    Inner Class
    A custom Thread class which runs until the mode is selected by the player which it then returns to the controller.

3. View

1. Class ModSelectionWindow: (extends JFrame)

Κλάση για την δημιουργία παραθύρου διαλόγου με τον χρήστη και επιστροφή της επιλογής αυτής σαν ένα ακέραιο (μεταξύ 0-3).

Πεδία:

- `private static final JPanel startingScreen = new JPanel();`
- `private static final JPanel radioButtons = new JPanel();`
- `private static final JPanel confirmCancel = new JPanel();`
- `static JFrame teliko ;`
- `private static JButton confirm;`
- `private static JButton cancel;`
- `private static JRadioButton reducedArmy;`
- `private static JRadioButton onlyForword;`
- `private int m_mode = -1;`

Μεθόδοι:

1. `public void init() {return m_mode;}`
   Transformer:
   Initializes the modSelectionWindow and adds components
2. `private void initTeliko() {return m_mode;}`
   Transformer:
   Initializes the frame (teliko) ,adds the background and components
3. `private JPanel setStartingScreen() {return m_mode;}`
   Transformer:
   Creates and returns a panel for the starting screen (first screen the user sees and adds the required UI)
4. `private JPanel setRadioButtons() {return m_mode;}`
   Transformer:
   Creates and returns a panel with radio buttons from which the user will select the mods
5. `private JPanel setConfirmButton() {return m_mode;}`
   Transformer:
   Creates and returns the panel with the confirm and cancel button
6. `private JPanel setTitle() {return m_mode;}`
   Transformer:
   Creates and returns the panel with the title of the game
7. `private void setBackground() {return m_mode;}`
   Transformer:
   Sets the background of the frame.

8. `public int getMode() {return m_mode;}`
   Accessor
   Returns the calculated mode (0: no mods, 1: reduced army, 2:no retreat, 3: both)

2. Class Field:

Κλάση για την υλοποίηση της διεπαφής (αριστερής) του πεδίου "μάχης" παίρνοντας πληροφορίες από το ταμπλό. Για κάθε παίκτη υπάρχει το δικό του JPanel το οποίο αποτελείται από ένα 8x10 GridLayout με JButtons τα οποία βλέπει μέσα από το reference στο board.

Πεδία:

- `private JPanel hiddenBlue;`
- `private JPanel hiddenRed;`
- `private Board f_board;`

Μεθόδοι:

1. `public void setHiddenBlue(){}`
   Transformer:
   Sets hidden blue panel, panel red player is seeing
2. `public void setHiddenRed(){}`
   Transformer:
   Sets hidden red panel, panel blue player is seeing
3. `public void swapFields(boolean turnBlue) throws PathNotFoundException {}`
   Transformer:
   Changes which players' panel is visible
4. `public JPanel getHiddenBlue(){}`
   Accessor:
   Returns the panel red player is seeing
5. `public JPanel getHiddenRed(){}`
   Accessor:
   Returns the panel blue player is seeing

3. Class Stats:

Κλάση για την υλοποίηση της διεπαφής (δεξιάς) των στατιστικών παίρνοντας πληροφορίες από τους παίκτες. Δημιουργούνται 2 τέτοια αντικείμενα, ένα για κάθε παίκτη τα οποία εμφανίζονται στον αντίστοιχό γύρο, με το να κρύβεται το άλλο και να γίνεται ορατό αυτό.

Πεδία:

- `private Player m_player;`

- private JCheckBox l_mod1;
- private JCheckBox l_mod2;
- private boolean mod1;
- private boolean mod2;
- private int win_rate;
- private int revives;
- private int round;
- private int capture1;
- private int capture2;
- private int capture3;
- private int capture4;
- private int capture5;
- private int capture6;
- private int capture7;
- private int capture8;
- private int capture9;
- private int capture10;
- private int totalCaptures;

Μεθόδοι:

1. private void activeMods(int mode){}
   Transformer:
   Generates the Active Mods JPanel to go into the final JFrame
2. private void Statistics(){}
   Transformer:
   Generates the Statistics JPanel to go into the final JFrame
3. private void captures(){}
   Transformer:
   Generates the Captured Units JPanel based on the player reference field to go into the final JFrame
4. public void update(){}
   Transformer:
   Changes this instance of Field and also adding all the components and setting the images on the buttons
5. public void nextTurn(int turn){}
   Transformer:
   Increments the round counter on the statistics JPanel
6. public void hideAll(){}
   Transformer:
   Sets all Panels that correspond to a certain player as not visible
7. public void showAll(){}
   Transformer:
   Sets all Panels that correspond to a certain player as visible
8. public void addComponents(JFrame frame){}

Accessor:
Adds all components of the Stat object to the given frame

4. Class View: (extends JFrame)

Η κύρια κλάση για την υλοποίηση της οπτικής διεπαφής με τον χρήστη.

Πεδία:

- `private Stats[] statScreen;`
- `private Field fieldScreen;`
- `private Controller controller;`
- `private Player p1;`
- `private Player p2;`
- `private ArrayList<JButton> graveyard;`
- `private int mode;`
- `private ModSelectionWindow msw;`
- `private JFrame gameView;`

Μεθόδοι:

1. `public void initView() {}`
   Transformer:
   Initializes the View Frame and adds the background
2. `public void updateView(boolean turnRed , int round) {}`
   Transformer:
   A method to swap the field visible to the next players field (JPanel) and also flip the cards of the current player and unflip the next ones cards
3. `private JPanel closeButton() {}`
   Transformer:
   Creates and returns to the object, a panel with a close button which terminates the application
4. `public void updateStats() {}`
   Transformer:
   Updates the statsBlue and statsRed panels each turn

5. Class reviveSelectionFrame: (extends JFrame)

Η κλάση που δίνει στον παικτη την επιλογή να επιλέξει ποιό από τα ranks των πιονιών του που έχει αιχμαλωτίσει ο αντίπαλος θέλει να διασώσει και το επιστρέφει στην κλάση Player από την οποία καλέστηκε.

Πεδία:

- `String s ;`
- `String imageURL;`

- `private int rank;`
- `private Player m_player;`

Μεθόδοι:

1. `public reviveSelectionFrame() {}`

   Constructor

   Creates a new Input Dialog Box which adds to this instance of JFrame and waits for input from the player as to the rank it wants to revive.
2. `public String choice(){}`

   Accessor

   Calculates if the player's choice is null and returns -1 or else returns the selected rank as string
3. `public int getRank(){}`

   Accessor

   Returns the choice as an integer.

6. Class LoadingScreen: (extends JFrame)

Η κλάση που χρησιμοποιείται για την μεταβίβαση από μενού σε παιχνίδι διότι το παιχνίδι χρειάζεται λίγη ώρα να δημιουργήσει όλα τα αντικείμενα.

Πεδία:

- `private JLabel loading;`
- `private JLabel background;`

Μεθόδοι:

1. `public void count() {}`

   Transformer:

   Adds a full stop at the end of the "Loading" text to show progression

7. Class winningFrame: (extends JFrame)

Η κλάση για τη δημιουργία του Frame του νικητή στο τέλος του παιχνιδιού

Πεδία:

- `private Player m_player;`

Μεθόδοι:

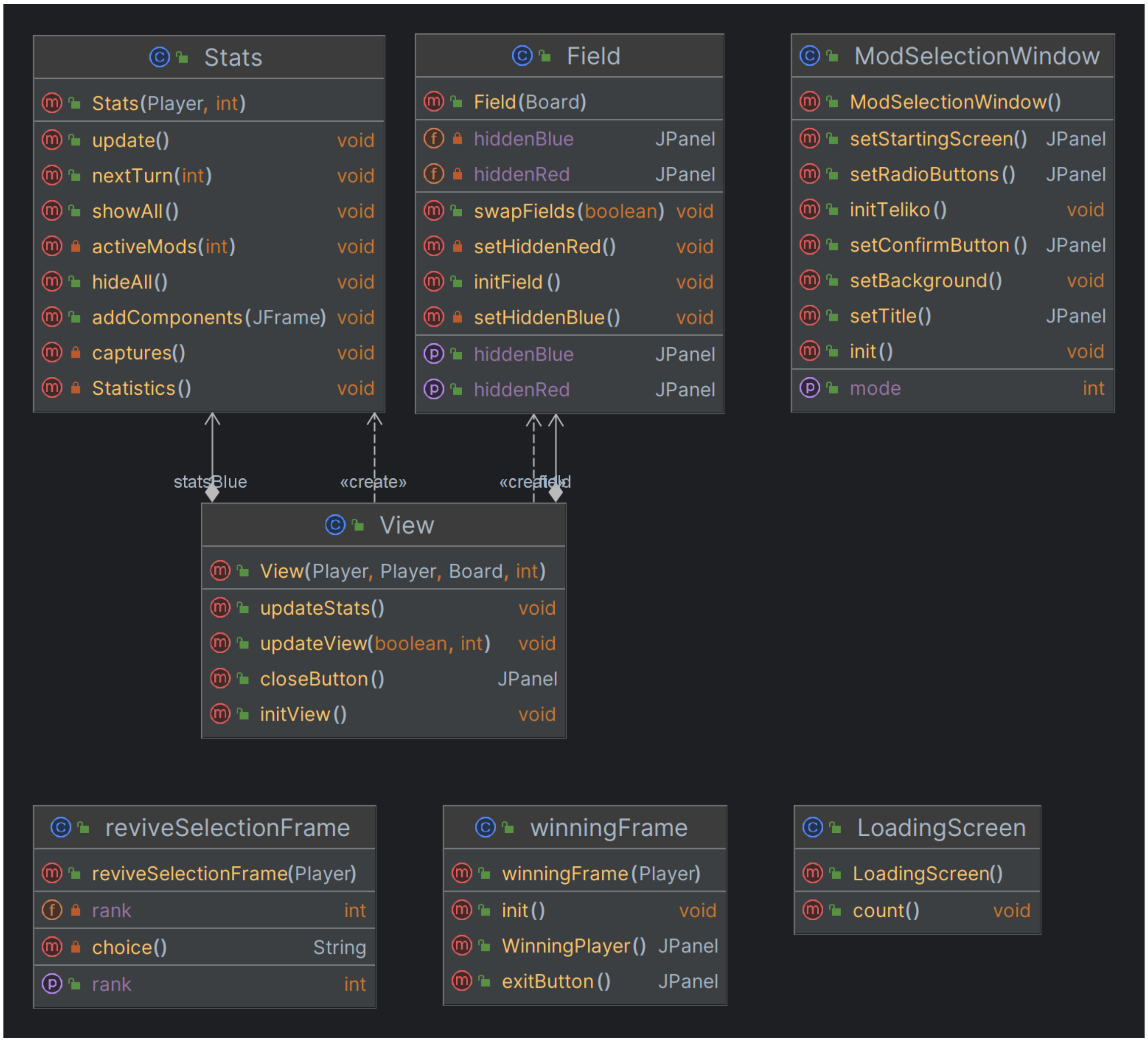
1. `private void init() {}`

   Transformer:

   Initializes the frame ,adds the background and components

2. `private JPanel WinningPlayer() {}`
   Transformer:
   Returns the panel with the winning player's name
3. `private JPanel exitButton() {}`
   Transformer:
   Returns the panel with the exit button

VIEW UML



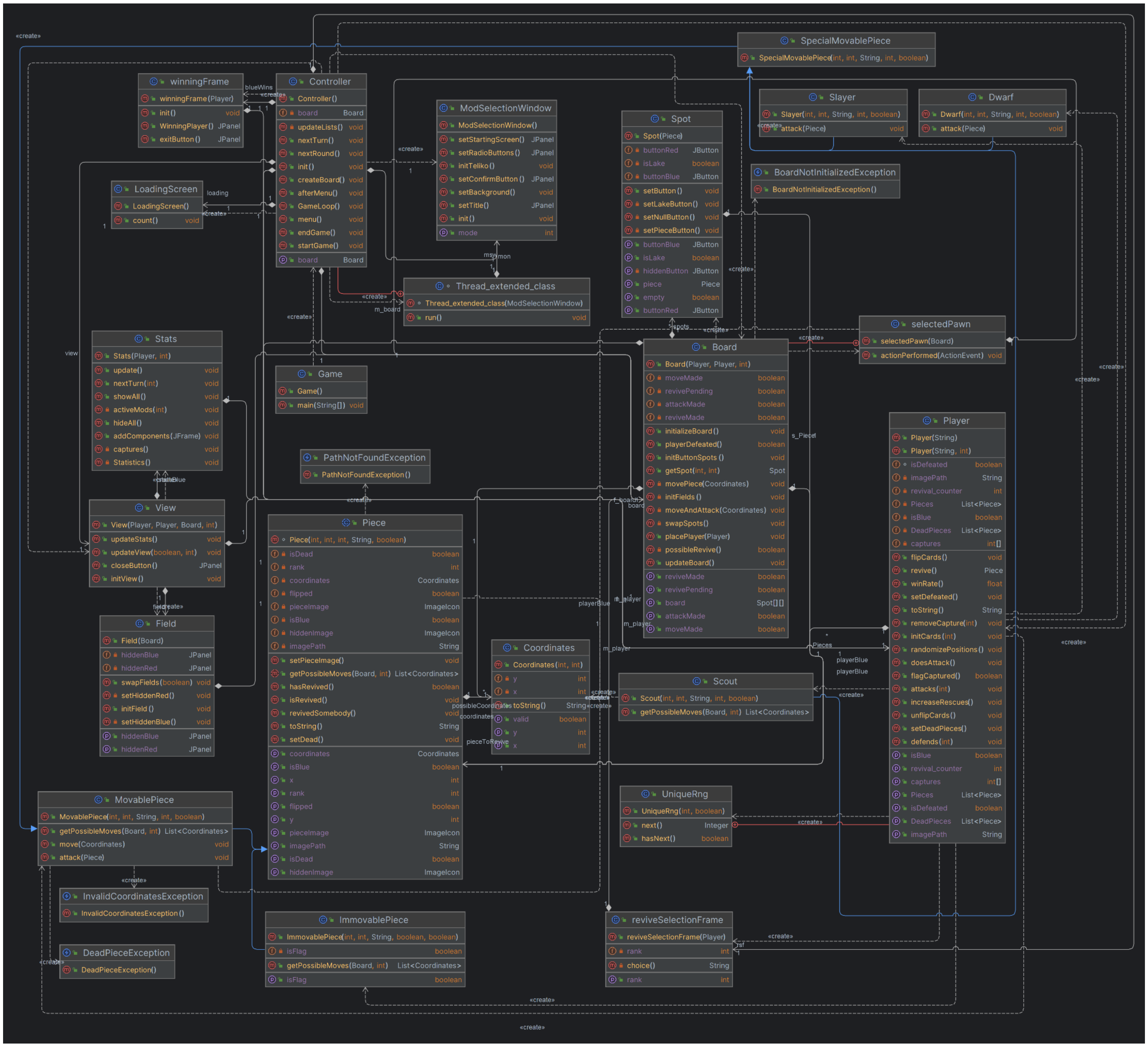
4. Game

1. Class Game:

This is the class used as the application by making an instance of the controller and starting the game.

COMPLETE UML

(Also attached as a .uml file)



3. Conclusion

This project took a lot of effort and through it I learned a great deal of game development mechanics and object oriented programming attributes that I never thought were possible. One example of this is the division of code in smaller segments to make it easier to understand and debug later. Even though I think my technique on the matter still needs a lot of improvements.

One more thing I would like to highlight is the amount of effort it takes to learn how to use Java awt and swing, as a true beginner on the matter and despite the final product I can think of a lot of optimisations that could take place to improve this project tremendously, such as the stuttering when the redrawing occurs and I would've liked to add a fight scene when pieces attack one another.

The biggest issue I faced was with consistency as I was not punctual and over the course of the last three weeks I was contradicting myself, with the logic I wanted to use, a lot of times. So I made it difficult for myself since a lot of different classes contained copies of the same objects instead of taking them as arguments for example. This is an area that only improves with interaction with such problems and this was a great opportunity to do so.

There are a lot of points that need improvement. For example the distinction between Model - View - Controller is not very clear, as it was not very clear when I began writing the code which then translates to my code not showing the distinction. The main problem is the somewhat fusion of the Model with the View in some cases, such as the Player creating the reviveSelectionFrame instead of the controller. There is also the issue that each Spot holds two JButtons each time so when updating the board each round the creation of needed objects takes a lot of time instead of just updating the image from the View.

In conclusion, this project was a lot of fun to work on and taught me a lot of things. I would gladly tackle such a project in the future.

Michalis Ierodiakonou
csd4773