Department Of Computer Science

Object - Oriented Programming

HY-252

Project 2023 - 2024

Sorry! Board Game

Phase B of the Project:

- Implementation of the application based on the ideas and principles of object-oriented programming.

Prepared by:

Marios Papadakis
AM: 5254

csd5254@csd.uoc.gr

January 17, 2024

TABLE OF CONTENTS

1. Introduction

The project will be implemented using the Model-View-Controller (MVC) design pattern. The MVC pattern separates an application into three main logical components: Model, View, and Controller. The Controller acts as an intermediary between the Model and View components, enabling the interconnection between them. In the report, we will analyze the important parts of the Model and Controller components that are relevant to this phase, and briefly discuss the View component.

2. The “Model” Package

The Model package includes various small classes housing essential game data and executing multiple functions. These components, when appropriately linked to the Controller package, constitute the core of the game at the memory level. This package consists of 10 classes for the Cards, 9 classes for the squares of the Deck, a class for the Deck itself, a class for the Pawns and lastly a class for the players.

Further Analyzing:

2.1 Card Directory

The card directory consists of an Abstract class of Card that models a card of the game and has some default methods and attributes. All of the other classes extend this or the NumberCard class that will be mentioned further down the report.

2.1.1 Abstract class Card.

The purpose of this class is to define the essential methods that all cards in the game must implement. It establishes a common set of functionalities that cards should have, including checking if a card has been played, activating and deactivating the cards, setting and retrieving card images, and determining if a pawn can be moved based on the card's properties. The class is designed to provide a standardized blueprint for various types of cards within a game, ensuring consistency in their behavior and allowing for seamless integration into a larger system.

Attributes:

- private boolean isPlayed;
  Description: This variable stores if the card has been played or not.

- private String cardImage;
  Description: This variable stores the image of the card.

- private String imagePath;
  Description: This variable stores the path of the image of the card.

- private transient Image image;
  Description: This variable stores the image of the card in order to save the game.

Methods:

- public void setImagePath(String imagePath);
  Description: This method checks if the card is played or not.
- public String getImagePath()
  Description: Gets the Path of the image of the card.

- public boolean cardIsPlayed();
  Description: This method Checks if the card has been played.

- public void cancelCard()
  Description: This method sets the card as played so that it can't be used again.
- public void activateCard();
  Description: Sets the card as not played so that it can be used again.

- public abstract void executeMove(Pawn pawn, Deck deck);
  Description: This method Moves the pawn according to the card's value.

- public abstract boolean canMove(Pawn pawn, Deck deck);
  Description: Checks if the Pawn can be moved by using the card.

- public String getImage()
  Description: This is a getter for the image of the card. It returns the image of the card.

- public void setImage(String Image)
  Description: This method is a setter for the image of the card.

- public void setActualImage(Image image);
  Description: Sets the Image of the card to the given one.

- public void sendToStartSquare(Pawn pawn, Deck deck);
  Description: Sets the Position of the Pawn to the Start Square.

2.1.2 Abstract class NumberCard

This abstract class extends the "Card" interface, providing a template for number cards in the game. It includes attributes for the card's number, played status, and an image. The constructor initializes the card with a given number and sets its played status to false. The class overrides methods from the interface to check if the card is played, set and retrieve descriptions, move a pawn based on the card's value, and manage the card's image. It serves as a foundation for specific number card implementations, ensuring consistent behaviors and facilitating code reuse in a card game model.

Attributes:

- private int number
  Description: Stores the number of the card.

Methods:

- public void setNumber(int number)
  Description: This method sets the number of the card to the given number.

- public int getNumber()
  Description: Returns the number of the card.

- public void executeMove(Pawn pawn, Deck deck);
  Description: This method Moves the pawn according to the card's value.

- private void handleSafetyZoneMove(Pawn pawn)
  Description: Handles the move of the pawn while in the safety zone.

- private void handleNonSafetyZoneMove(Pawn pawn, Deck table)
  Description: Handles the move of the pawn while not in the safety zone.

- public void handleSlides(Pawn pawn, Deck table, int originalPosition)
  Description: Handles the slides of the pawn. Used in the executeMove method.

- private void handleYellowSlides(Pawn pawn, Deck table, int originalPosition)

Description: Handles the slides of the yellow pawn.

- private void handleRedSlides(Pawn pawn, Deck table, int originalPosition)
  Description: Handles the slides of the red pawn.

- private int calculateTargetPosition(Pawn pawn)
  Description: Returns the target position that the Pawn will be moved to. Used in the handleNonSafetyZoneMove method.

- private boolean isInSafetyZone(Pawn pawn)
  Description: Returns true if the pawn is in the safety zone, false otherwise.

- public boolean canMove(Pawn pawn, Deck table)
  Description: Returns true if the pawn can be moved by using the card, false otherwise.

- private boolean isValidMoveWithinSafetyZone(Pawn pawn, Deck table, int targetPosition)
  Description: Checks if the move is valid within the safety zone.

- private int calculatePossiblePosition(Pawn pawn)
  Description: Calculates the possible position that the Pawn will be moved to.

2.1.3 Class SimpleNumberCard

The "SimpleNumberCard" class extends the "NumberCard" abstract class, focusing on simple number cards in the card game. These cards don't have special abilities like moving a Pawn Counter-clockwise. They only move a Pawn depending on the Value of the Card. This class includes attributes for played status and description. The constructor initializes the card with a given number.

Methods:

public SimpleNumberCard(int number)
Description: This is a constructor that Creates a new instance of a simple card (3, 5, 6, 8, 9, 12). It takes as an argument the number of the card that is going to be created.

2.1.4 Class NumberOneCard

The "NumberOneCard" class extends the "NumberCard" abstract class, representing the Number One card in a card game. It specifies that the card moves the pawn one space forward and can also

move from the start. The class includes attributes for played status and description, and the constructor initializes the card with the number 1.

Methods:

- public NumberOneCard()
  Description: This is a constructor that creates a new Number 1 card.

- public void executeMove(Pawn pawn, Deck table)
  Description: Moves the Pawn 1 space forward, clockwise.

- public boolean canMove(Pawn pawn, Deck table)
  Description: Returns true if the pawn can be moved by using the card, false otherwise.

- private boolean checkSquare(Square square, PAWNS_COLOR color)
  Description: Checks if there is a Pawn in the Square that is the same color as the Pawn that will be moved.

- public String toString()

2.1.5 Class NumberTwoCard

The "NumberTwoCard" class extends the "NumberCard" abstract class, representing a Number Two card in a card game. This card can move the pawn two spaces forward, clockwise, and allows the player who draws it to play again. The class includes attributes for played status and description, and the constructor initializes the card with the number 2.

Methods:

- public NumberTwoCard()
  Description: This is a constructor that creates a new Number 2 card.

- public boolean canMove(Pawn pawn, Deck table)
  Description: Returns true if the pawn can be moved by using the card, false otherwise.

- public void executeMove(Pawn pawn, Deck table)
  Description: Moves the card 2 spaces forward, can Move from the Start.

- private boolean isMoveAllowed(Square targetSquare, PAWNS_COLOR colorToCheck)
  Description: Checks if the move is allowed. This method is used in the canMove method.

- public String toString()

2.1.6 Class NumberFourCard

The "NumberFourCard" class extends the "NumberCard" abstract class, representing a Number Four card in a card game. This card can move the pawn four spaces backward. The class includes attributes for played status and description, and the constructor initializes the card with the number 4.

Methods:

- public NumberFourCard()
  Description: This is a constructor that creates a new Number 4 card.

- public boolean canMove(Pawn pawn, Deck table)
  Description: Checks if the Pawn can be moved by using the card.

- public void executeMove(Pawn pawn, Deck table)
  Description: Moves the Pawn 4 spaces backwards.

- public String toString()

2.1.7 Class NumberSevenCard

The "NumberSevenCard" class extends the "NumberCard" abstract class, representing a Number Seven card in a card game. This card can move the pawn seven spaces clockwise or split the move between two pawns. The class includes attributes for played status and description, and the constructor initializes the card with the number 7.

Methods:

- public NumberSevenCard()
  Description: This is a constructor that creates a new Number 7 card.

- public void executeMove(Pawn pawn1, int move1, Pawn pawn2, int move2, Deck table)
  Description: This method splits the move between the 2 Pawns.

- public void executeMove(Pawn pawn, int move, Deck table)
  Description: Moves the Pawn individually by move spaces.

- private void handleRegularMove(Pawn pawn, int move, Deck table)
  Description: Moves the Pawn individually by move spaces.

- private void handleSafetyZone(Pawn pawn, int move)
  Description: Handles the move is the pawn is in the safety zone.

- private void handleRedPawnMove(Pawn pawn, int move, Deck table)
  Description: Move the Red pawn by move spaces, depending on the pawn's position.

- private void handleYellowPawnMove(Pawn pawn, int move, Deck table)
  Description: Move the Yellow pawn by move spaces, depending on the pawn's position.

- private void handleNormalMove(Pawn pawn, int move, Deck table)
  Description: Move the pawn by move spaces.

- private void handleSpecialMoveRedStart(Pawn pawn, int move)
  Description: Handle the special move of the Red pawn if it is on the start square.

- private void handleSpecialMoveYellowStart(Pawn pawn, int move)
  Description: Handle the special move of the Red pawn if it is on the start square.

- public boolean canMove(Pawn pawn1, int move1, Pawn pawn2, int move2, Deck table)
  Description: Returns true if the pawn can be moved by move1 and move2 spaces.

- private int calculatePosition(Pawn pawn, int move)
  Description: Calculates the possible position of the pawn.

- public boolean canMove(Pawn pawn, int move, Deck table)
  Description: Returns true if the pawn can be moved by move spaces.

- private boolean isMoveValid(Pawn pawn, int move, Deck table, int destination)
  Description: Checks if the move is valid.

- private int calculatePossiblePosition(Pawn pawn, int move)
  Description: Calculates the possible position of the pawn.

- public String toString()

2.1.8 Class NumberTenCard

The "NumberTenCard" class extends the "NumberCard" abstract class, representing a Number Ten card in a card game. This card can move the pawn ten spaces clockwise or move one space backward.

Methods:

- public NumberTenCard()
  Description: This is a constructor that creates a new Number 10 card.

- public void movePawnBackwards(Pawn pawn, Deck table)
  Description: Moves the Pawn 1 space backwards.

- private int calculateNewPosition(Pawn pawn, int currentPosition) Description: Calculates the new position of the Pawn.

- public boolean canMoveBackwards(Pawn pawn, Deck deck)
  Description: Checks if the Pawn can be moved by using the card.

- public String toString()

2.1.9 Class NumberElevenCard

The "NumberElevenCard" class extends the "NumberCard" abstract class, representing a Number Eleven card in a card game. This card can move the pawn eleven spaces clockwise or swap the position of one of the player's pawns with another player's pawn.

Methods:

- public NumberElevenCard()
  Description: This is a constructor that creates a new Number 11 card.

  public void executeMove(Pawn pawn, Pawn opponentPawn, Deck table)
  Description: Moves the Pawn by swapping positions with the opponentPawn.

- private void swapPositions(Pawn pawn1, Pawn pawn2, Deck table)
  Description: Responsible for swapping the positions of the two pawns.

- public boolean canMove(Pawn pawn, Pawn opponentPawn, Deck table) Description: Checks if the Pawn can be moved by using the card.

- private boolean isRestrictedSquare(Square square)
  Description: Checks if the Square is a Start, Home or Safety Zone Square.

- public String toString()

2.1.10 Class SorryCard

The "SorryCard" class extends the "Card" abstract class and represents a Sorry Card in a card game. This card can be played and perform a specific move that involves swapping the positions of the player's pawn and the opponent's pawn under certain conditions. The constructor initializes the card with the played status set to false.

Methods:

- public boolean canMove(Pawn pawn, Pawn oppPawn, Deck deck) Description: Checks if the Pawn can be moved by using the card.

- public void executeMove(Pawn pawn, Pawn opponent, Deck deck) Description: Swaps the Pawn (that is in a Start square) with an Opponents Pawn that is not in a Safety Zone.

- private void handleSlides(Pawn pawn, Deck table, int originalPosition)
  Description: This is a method is responsible for handling the slides for the Sorry Card.

- private void handleYellowSlides(Pawn pawn, Deck table, int originalPosition)
  Description: This is for managing the Slides of a Yellow Pawn.

- private void handleRedSlides(Pawn pawn, Deck table, int originalPosition)
  Description: This is for managing the Slides of a Red Pawn.

- public void executeMove(Pawn pawn, Deck deck)
  Description: This does nothing but I had to implement it because of the super class. ☺

- public boolean canMove(Pawn pawn, Deck deck)
  Description: Same here ☺

- public String toString()
  Description: You know what this does…

2.2 Square Classes

2.2.1 Abstract class Square

The "Square" class is an abstract class representing a square on the game board. It serves as the superclass for various types of squares, including EndSlideSquare, EndSquare, HomeSquare, SlideSquare, and StartSquare. The class contains attributes for the position and color of the square etc. and it provides methods for checking if the square is occupied, setting and retrieving the position, and setting and retrieving the color.

Attributes:

- private int position
  Description: This is a variable that stores the Position of the Square.

- private Color color
  Description: Stores the color of the Square.

- private Pawn occupied;

Description: Stores the Pawn that occupies the Square.

Methods:

- public boolean isOccupied()
  Description: Returns if the Square is occupied with a Pawn or not.

- public Pawn getIsOccupied()
  Description: Returns the Pawn that occupies the Square.

- public void setOccupied(Pawn pawn)
  Description: Sets the Square as occupied with the given Pawn.

- public void setPosition(int position)
  Description: Sets the position of the Square to the given one.

- public int getPosition()
  Description: Returns the position of the Square.

- public void setColor(Color color)
  Description: Sets the color of the Square to the given one.

- public Color getColor()
  Description: Returns the color of the Square.

2.2.2 Class StartSquare

The "StartSquare" class is a subclass of the "Square" abstract class and represents a square on the game board that serves as the starting point for a particular player.

Methods:

- public StartSquare(int position, Color color)
  Description: This is a constructor for the Start Square, it creates a new Start Square.

2.2.3 Class SimpleSquare

The "SimpleSquare" class is a subclass of the "Square" abstract class and represents a basic square on the game board without any special effects. It is a straightforward implementation with a constructor that initializes the simple square with a given position.

Methods:

- public SimpleSquare(int position, Color color)
  Description: This is a constructor for the Simple Square, it creates a new Simple Square.

2.2.4 Class SafetyZoneSquare

The "SafetyZoneSquare" class is a subclass of the "Square" abstract class, representing a square on the game board that serves as a safety zone and can only be accessed by a specific player.

Methods:

- public SafetyZoneSquare(int position, Color color)
  Description: This is a constructor for the SafetyZone Square, it creates a new Safety Zone Square.

2.2.5 Class HomeSquare

The "HomeSquare" class is a subclass of the "Square" abstract class, representing the home square of a player on the game board.

Methods:

- public HomeSquare(int position, Color color)
  Description: This is a constructor for the Home Square, it creates a new Home Square.

2.2.6 Class StartSlideSquare

The "StartSlideSquare" class is a subclass of the "Square" class, representing a square on the game board that serves as the starting point of a slide. This class extends the functionality of the "Square" class, indicating that it is specifically a starting point for a slide.

Methods:

- public StartSlideSquare(int position, Color color)
  Description: This is a constructor that instantiates a new Start Slide Square

2.2.7 Class InternalSlideSquare

The "InternalSlideSquare" class is a subclass of the "Square" class, representing a square on the game board that is part of a slide but is neither the starting point nor the ending point.

Methods:

- public InternalSlideSquare(int position, Color color)
  Description: This is a constructor that instantiates a new Internal Slide Square

2.2.8 Class EndSlideSquare

The "EndSlideSquare" class is a subclass of the "Square" class, representing a square on the game board that marks the end of a slide. The purpose of having specific subclasses like "StartSlideSquare," "InternalSlideSquare," and "EndSlideSquare" is to distinguish different types of squares within a slide, each serving a specific role in the game.

Methods:

- public EndSlideSquare(int position, Color color)
  Description: This is a constructor that instantiates a new End Slide Square

2.3 Deck Class

The Deck class is responsible for the initialization of the Deck of cards, and it consists of the main board with the squares and the Pawns of the players. It incorporates methods to initialize the cards, board, and pawns, as well as shuffle the deck and draw the top card.

Attributes:

- private Card currentCard
  Description: This is a variable that stores the CurrentCard that is being played.

- private ArrayList<Square> squares
  Description: This is an ArrayList that stores the squares of the Board.

- ArrayList<Card> gameCards = new ArrayList<>()
  Description: This is an ArrayList that stores the game cards of the Deck.

- int cardsLeft;
  Description: This is an variable that stores how many cards are left before shuffling.

Methods

- public ArrayList<Card> getGameCards()
  Description: This is a getter method that returns the gameCards ArrayList.

- public ArrayList<Square> getBoard()
  Description: This is a getter method that returns the squares of the board.

  public void setGameCards(ArrayList<Card> gameCards)
  Description: This is a method to set the Game Cards to the ArrayList

- public void initDeck()
  Description: This is a method that Initializes the Cards and the Board.

- public void initCards()
  Description: This is a method to initialize the Cards for the game.

- public void initBoard()
  Description: This is a method to initialize the square for each player.

- public Card drawCard()
  Description: This is a method used to Draw a Card from the Top of the Deck.

- public void shuffleCards()
  Description: This is a method used to shuffle the cards of the Deck if the Deck gets empty.

- private static Card getCard(int[] cardNumbers, int j, String[] cardImages)
  Description: This is a method used to get the card with index j set its Image and return it.

- public Card getCurrentCard()
  Description: This is a method used to return the current card.

- public int getCardsRemaining()

Description: This is a method used to return how many cards remain before shuffling the deck of cards.

2.4 Pawn Class

The "Pawn" class represents a Pawn piece with attributes such as color, position, activity status, and home status. It includes methods for accessing and modifying pawn properties, allowing for setting color, position, activity, and home status. This class encapsulates the essential features of a pawn in the game.

Attributes:

- PAWNS_COLOR PAWNSColor
  Description: This is an Enum that represents the color of the Pawn.

- private int position
  Description: This is a variable that will store the position of the Pawn.

- private boolean isActive
  Description: This is a boolean variable that will tell us if the Pawn is active or not.

- private boolean isAtHome = false;
  Description: This is a boolean variable that will tell us if the Pawn is at Home or not. This variable is first set to false since the Pawn is located in the Start Square.

Methods:

- public Pawn(PAWNS_COLOR PAWNSColor, int position)
  Description: Instantiates the Pawn with the given color. The position of the Pawn is set to 0 and it is set as active.

- public PAWNS_COLOR getColor()
  Description: This is a getter Method that returns the color of the Pawns.

- public void setColor(PAWNS_COLOR PAWNSColor)
  Description: This is a setter method that is used to set the colors to the desired one.

- public void setPosition(int position)
  Description: This is a setter method that is used to set the position of the Pawn.

- public int getPosition()
  Description: This is a getter method used to return an integer with the position of the Pawn in the Board.

- public boolean isActive()
  Description: This is a method that will return if the Pawn is active or not.

- public void setActive()
  Description: This is a setter method that is used to set the Pawn to be Active.

- public void setInactive()
  Description: This is a setter method that is used to set the Pawn to be InActive.

- public boolean isAtHome ()
  Description: This is a getter method that returns if the Pawn is located at the Home Square.

- public void setAtHome(boolean isAtHome)
  Description: This is a method that sets the Pawn to be at Home.

2.5 Player Class

The "Player" class represents a participant in the board game with attributes such as pawn color, player name, an array of pawns, and a turn indicator. It includes methods for accessing and modifying player properties, checking and setting the player's turn, and overriding the "equals" method for comparison. This class encapsulates the essential features of a player in the board game.

Attributes:

- PAWNS_COLOR PAWNSColor
  Description: This is an enumeration that holds all the Colors of the Players.

- private String playerName
  Description: This is String variable that holds the name of the player.

- Pawn[] pawns = new Pawn[2]
  Description: This is an array of 2 Pawn instances.

- private boolean playerTurn
  Description: This is a boolean variable that will represent if it's the turn for the player to play.

Methods:

- public Player(PAWNS_COLOR PAWNSColor, String playerName, Pawn[] pawns)
  Description: This is a constructor for the Player.

- public void setPawns(Pawn[] pawns)
  Description: This is a setter method that sets the Pawns of the Player.

- public Pawn[] getPawns()
  Description: This is a getter method to return the Pawns of the Player.

- public PAWNS_COLOR getColor()
  Description: This is a getter method to return the color of the Pawns.

- public void setColor(PAWNS_COLOR PAWNSColor)
  Description: This is a setter method to set the color of the Players Pawns.

- public void setPlayerName(String playerName)
  Description: This is a setter method to set the name of the Player.

- public String getPlayerName()
  Description: This is a getter method to return the player's name.

- public boolean getTurn()
  Description: This is a getter method that returns if it's the players turn to play.

- public boolean equals(Object obj)
  Description: This is a method that Overrides the equals method and it is used to compare two Player Objects for equality.

- public boolean isWinner()
  Description: This is a getter method that returns if the player is the winner or not.

- public Pawn getPawn(int i)
  Description: This is a getter method that returns the Pawn i of the player.

- public void swapTurns()
  Description: This is a method that swaps turns for the players.

- public String toString()
  Description: The string bro.

3. The “Controller” Package

The "Controller" package serves as the central component that coordinates interactions between the game model (represented by the Deck, Player, and other game elements) and the user interface (View). It manages the initialization of the game, including players, the deck, and the view. The class includes methods for initializing cards, setting listeners for various game elements, and determining the game's outcome, such as whether it has finished and who the winner is. The class also includes nested MouseListener classes for handling user interactions with cards, the fold button, and the game menu. This design allows for a separation of concerns, improving the modularity and maintainability of the code. Also it include the CotrollerTest class responsible for the Junit Tests as well as the GameState class that is responsible for saving the current game state into a .ser file.

3.1 Controller Class

Attributes:

- Deck board
  Description: This is a Deck variable that represents the deck of cards, the board and the Pawns.

- View view
  Description: This is the View variable that represents the view of the Game.

- Player player1_r
  Description: This is a Player variable that represents the Player 1 of the game.

- Player player2_y
  Description: This is a Player variable that represents the Player 2 of the game.

- private Player PlayerPlaying
  Description: This is a variable that stores which player is currently playing.

Methods:

- public void initGame()
  Description: This is a method that initializes the game.

- public void initBoardAndView()
  Description: Initializes the board and the view.

- public void initPlayers()
  Description: This is a method that initializes the Players.

- private void startGame()
  Description: This is a method that starts the game.

- public void updateInfo()
  Description: This is a method updates the info box.

- public void foldCards()
  Description: This is a method that folds the cards.

- public void swapTurns()
  Description: This is a method that swaps the turns of the Players.

- public void setListeners()
  Description: This is a method that sets the listeners.

- public Player getPlayerPlaying()
  Description: This is a method that gets the Player that is currently playing.

- public Object getBoard()
  Description: This is a method that gets the Board.

- public Object getView()
  Description: This is a method that gets the View.

- public Object getPlayer1_r()
  Description: This is a method that gets the first Player.

- public Object getPlayer2_y ()

Description: This is a method that gets the second Player.

- public void gameplay()
  Description: This is a method that simulates the game being played by 2 computers. It is used for the Junit test.

- public void autoPlay()
  Description: This is a method that simulates the game being played by 2 computers. It is used for the Junit test.

- private boolean GameIsFinished()
  Description: This is method that returns if the Game has finished or not.

- public static void NewGame()
  Description: This is a method that creates a new Game. This is used in the Menu bar in the button that says “New Game”.

- public void saveGame()
  Description: This is a method that saves the current state of the Game.

- public void loadGame()
  Description: This is a method that Loads the .ser file and updates the view in order to continue the saved game state.

- private Image loadImage(String imagePath)
  Description: This is a method that loads the Image with path = imagePath.

- private void printVictoryWindow(String string)
  Description: This is a method that pops up a window with the winning message.a

Inner Classes:

- private class cardButtonListener implements MouseListener
  Description: This is a Mouse Listener for the Card Button in order to draw a new card.

- private class menuBarListener implements MouseListener
  Description: This is a Mouse Listener for the MenuBar.

- private class menuBarListener implements MouseListener
  Description: This is a Mouse Listener for the MenuBar.

- private class pawnButtonsListener implements MouseListener
  Description: This is a Mouse Listener for the Pawn Buttons in order to choose a Pawn that the move will be applied on.

3.2 ControllerTest Class

The ControllerTest Class is a class that is responsible for running the Junit tests.

There are 3 different Tests that I made.

1. "testInitGame" This @test checks if everything is being Inititalized correctly.
2. "testSwapTurns" This @test checks if the Turns are being swaped correctly.
3. Lastly the "testGameplay" test is a test that runs the gameplay method from the controller and checks if the game Auto-plays correctly.

3.3 GameState Class

This Class contain some attributes for the players, the board and the player that currently plays. There is a constructor and some getters for all the attributes. This class is responsible for holding all the information in order to save the Game in a .ser file.

4. The "View" Package

The "View" class encapsulates the graphical user interface (GUI) for the board game, responsible for presenting the game elements and facilitating user interaction. Leveraging Java's Swing toolkit, it initializes and manages components such as buttons, labels, and menus. The GUI's layout, starting units, and information boxes are set up through methods like "initComponents" and "Start."

The class handles the visual representation of the game board, including pawn movements and player information updates. This class is responsible for creating the Fold Button, the Card Button and all the Pawn buttons and set them in the correct positions. Also the infoBox with its setters and

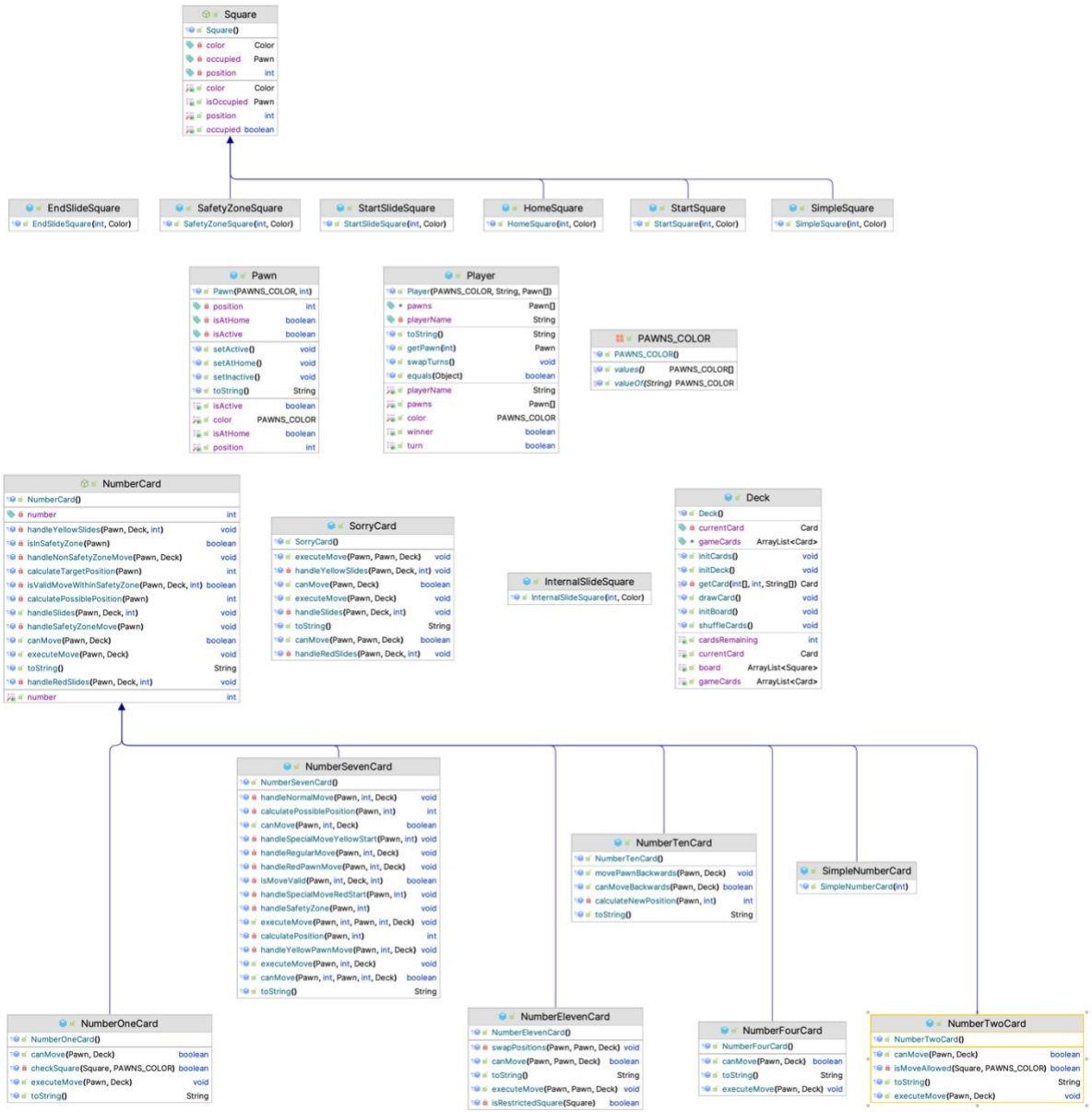
getters, and it also has an “repaintInfoBox” method in order to update the infoBox with any changes.

There is an updatePawns method that updates the Pawns view after any changes have been made. There are also setters and getters for the Pawns. There is a method that initializes all the squares correctly in order to form the board. A method that adds a background etc. There are more methods like the sendToStart and sendToHome and other helper methods.

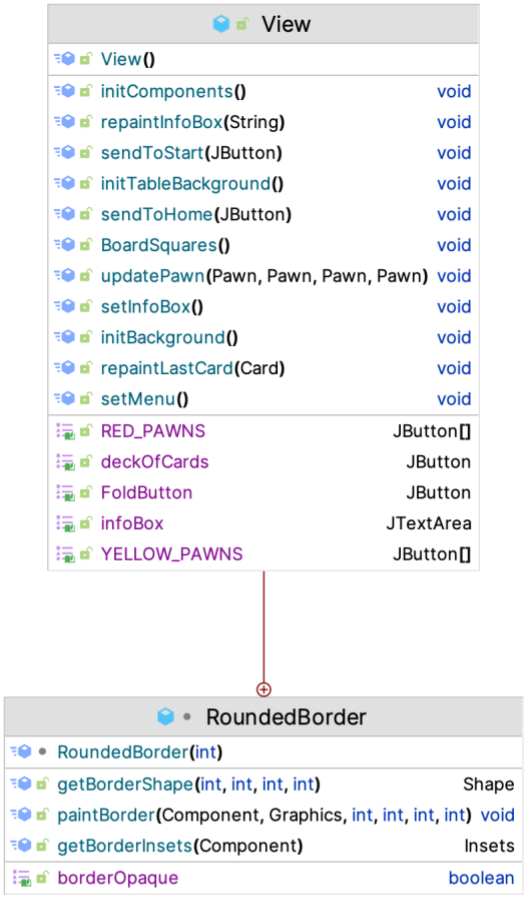
All in all the view class handles everything that the user can see. and it is responsible for creating the visual representation of what the controller does behind the scenes.

5. Interaction between classes - UML diagrams

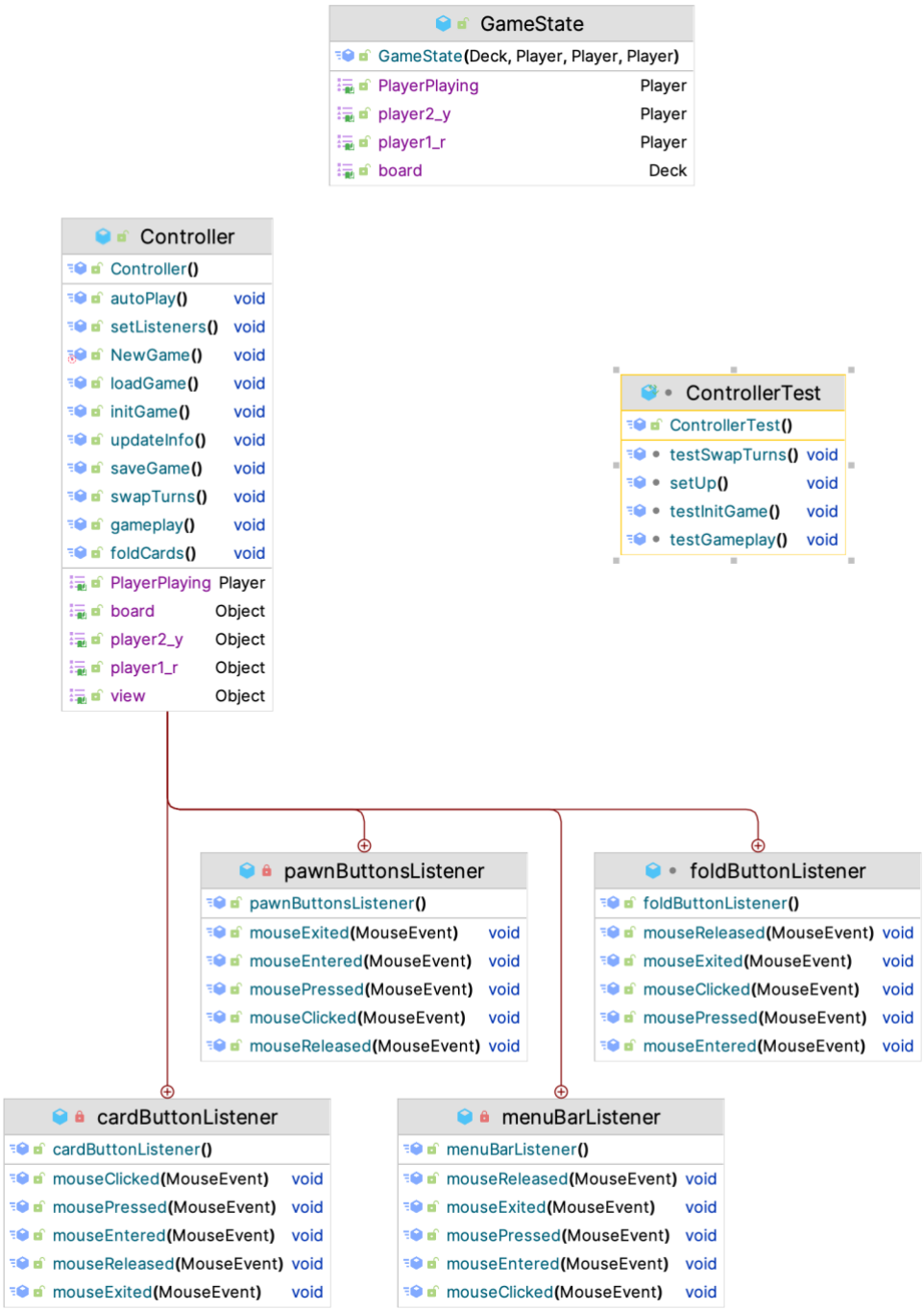
Model UML Diagram



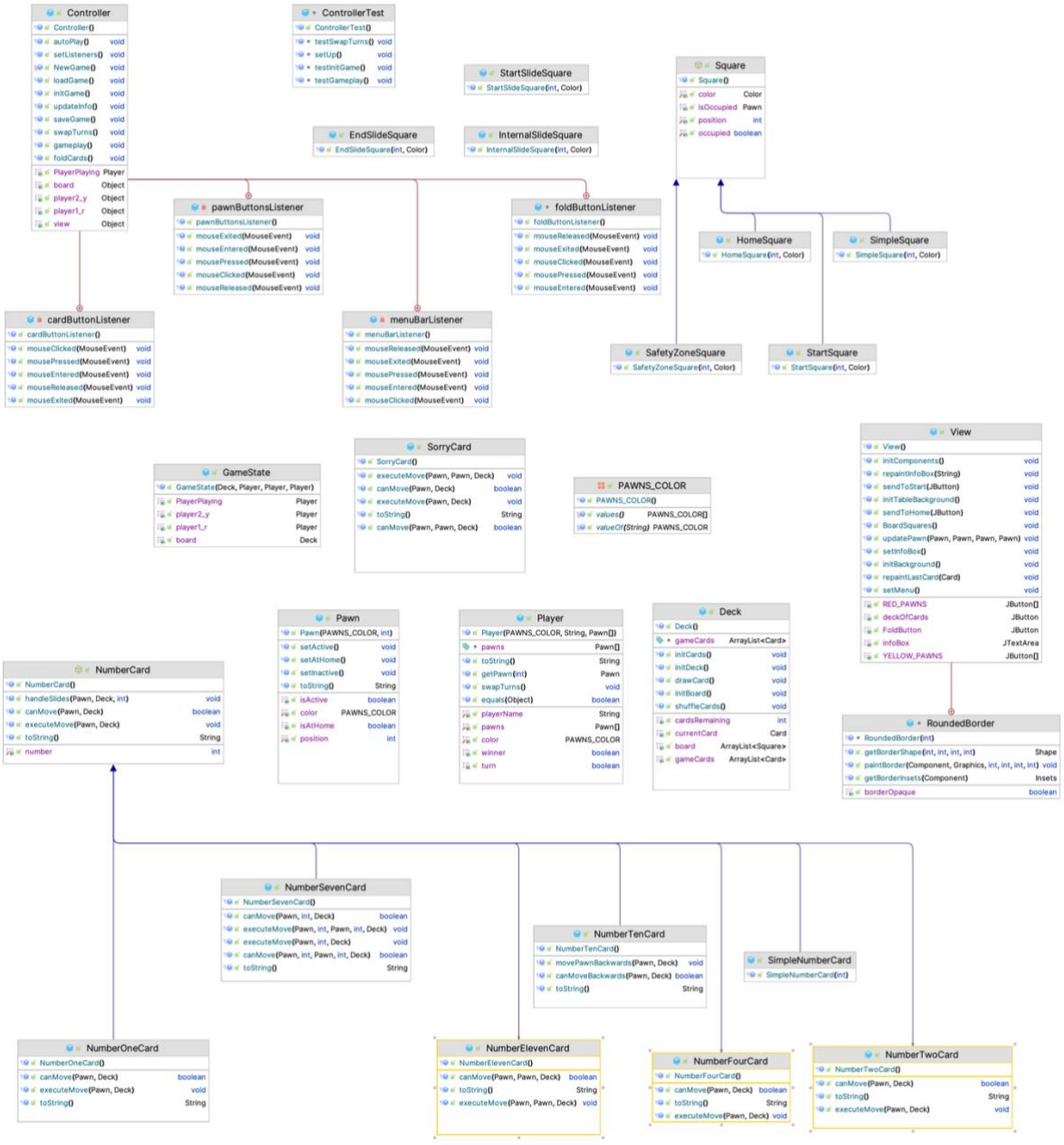
View UML Diagram



Controller UML Diagram



MVC UML Diagram



6. Functionality (Phase B)

After Completing the Second Phase of the project I was able to do everything smoothly. I also did the Bonus task for Saving the game state.

7. Conclusions

All in all it was a great project, I didn't face anything unusual or a major bug. I enjoyed it and if I had more time I would try to make the Game possible to play with 4 Players. The only problem I faced in the beginning was how to Start the whole project because we didn't have a lot of help. But I figured it out eventually.