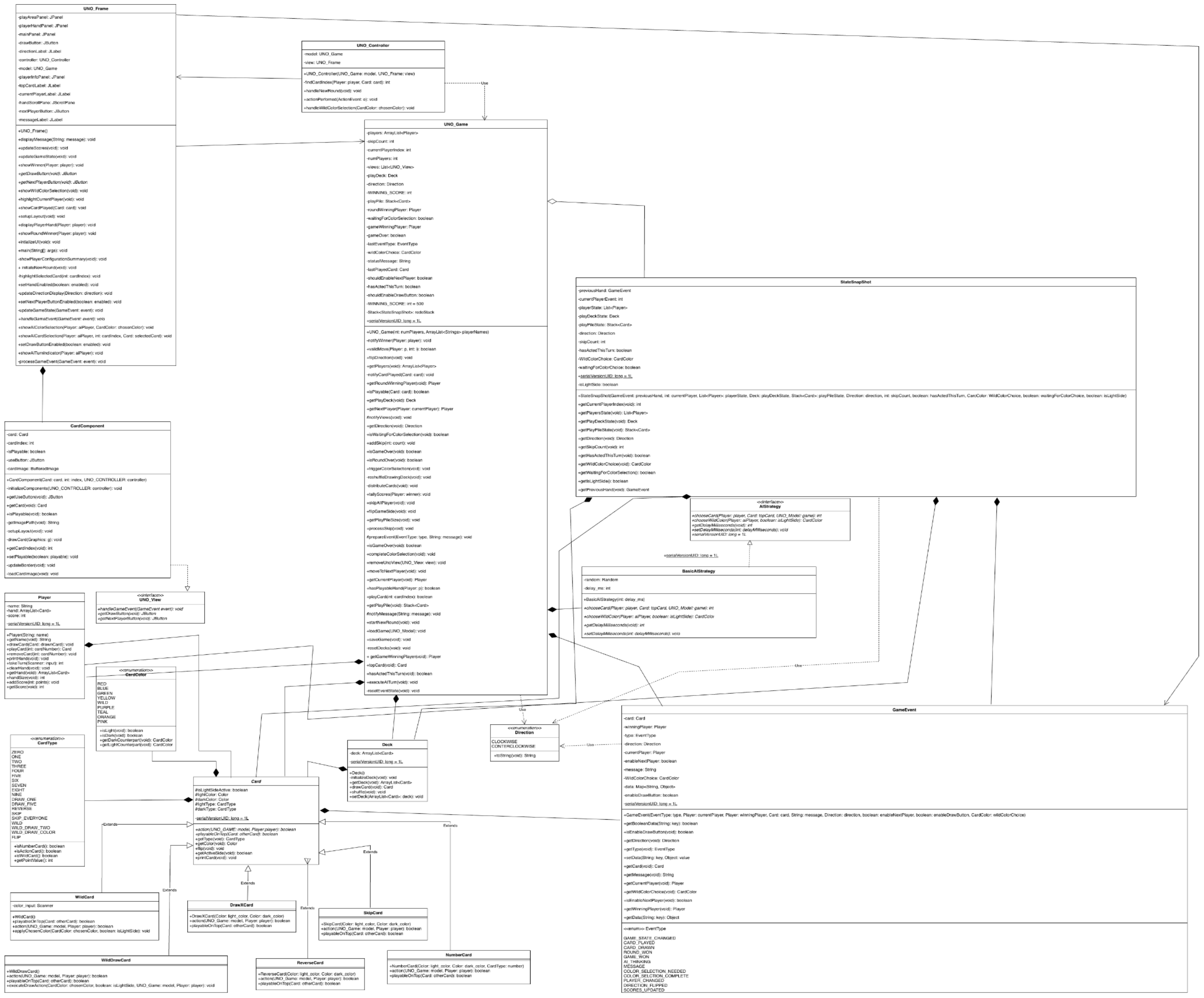
Uml Diagram:


UNO_Frame
-playAreaPanel: JPanel
-playerHandPanel: JPanel
-mainPanel: JPanel
-drawButton: JButton
-directionLabel: JLabel
-controller: UNO_Controller
-model: UNO_Game
-playerInfoPanel: JPanel
-topCardLabel: JLabel
-currentPlayerLabel: JLabel
-handScrollPane: JScrollPane
-nextPlayerButton: JButton
-messageLabel: JLabel
+UNO_Frame()
+displayMessage(String: message): void
+updateScores(void): void
+updateGameState(void): void
+showWinner(Player: player): void
+getDrawButton(void): JButton
+getNextPlayerButton(void): JButton
+showWildColorSelection(void): void
+highlightCurrentPlayer(void): void
+showCardPlayed(Card: card): void
+setupLayout(void): void
+displayPlayerHand(Player: player): void
+showRoundWinner(Player: player): void
+intializeUI(void): void
+main(String[]: args): void
-showPlayerConfigurationSummary(void): void
+ initiateNewRound(void): void
-highlightSelectedCard(int: cardIndex): void
+setHandEnabled(boolean: enabled): void
-updateDirectionDisplay(Direction: direction): void
+setNextPlayerButtonEnabled(boolean: enabled): void
-updateGameState(GameEvent: event): void
+handleGameEvent(GameEvent: event): void
+showAIColorSelection(Player: aiPlayer, CardColor: chosenColor): void
+showAICardSelection(Player: aiPlayer, int: cardIndex, Card: selectedCard): void
+setDrawButtonEnabled(boolean: enabled): void
+showAITurnIndicator(Player: aiPlayer): void
-processGameEvent(GameEvent: event): void
UNO_Controller
-model: UNO_Game
-view: UNO_Frame
+UNO_Controller(UNO_Game: model, UNO_Frame: view)
-findCardIndex(Player: player, Card: card): int
+handleNewRound(void): void
+actionPerformed(ActionEvent: e): void
+handleWildColorSelection(CardColor: chosenColor): void
Use
UNO_Game
-players: ArrayList<Player>
-skipCount: int
-currentPlayerIndex: int
-numPlayers: int
-views: List<UNO_View>
-playDeck: Deck
-direction: Direction
-WINNING_SCORE: int
-playPile: Stack<Card>
-roundWinningPlayer: Player
-waitingForColorSelection: boolean
-gameWinningPlayer: Player
-gameOver: boolean
-lastEventType: EventType
-wildColorChoice: CardColor
-statusMessage: String
-lastPlayedCard: Card
-shouldEnableNextPlayer: boolean
-hasActedThisTurn: boolean
-shouldEnableDrawButton: boolean
-WINNING_SCORE: int = 500
-Stack<StateSnapShot>: redoStack
+serialVersionUID: long = 1L
+UNO_Game(int: numPlayers, ArrayList<Strings> playerNames)
-notifyWinner(Player: player): void
+validMove(Player: p, int: i): boolean
+flipDirection(void): void
+getPlayers(void): ArrayList<Player>
-notifyCardPlayed(Card: card): void
+getRoundWinningPlayer(void): Player
+isPlayable(Card: card): boolean
+getPlayDeck(void): Deck
+getNextPlayer(Player: currentPlayer): Player
#notifyViews(void): void
+getDirection(void): Direction
+isWaitingForColorSelection(void): boolean
+addSkip(int: count): void
+isGameOver(void): boolean
+isRoundOver(void): boolean
+triggerColorSelection(void): void
-reshuffleDrawingDeck(void): void
-distributeCards(void): void
+tallyScores(Player: winner): void
+skipAllPlayer(void): void
+flipGameSide(void): void
+getPlayPileSize(void): void
+processSkip(void): void
#prepareEvent(EventType: type, String: message): void
+isGameOver(void): boolean
+completeColorSelection(void): void
+removeUnoView(UNO_View: view): void
+moveToNextPlayer(void): void
+getCurrentPlayer(void): Player
+hasPlayableHand(Player: p): boolean
+playCard(int: cardIndex): boolean
+getPlayPile(void): Stack<Card>
#notifyMessage(String: message): void
+startNewRound(void): void
+loadGame(UNO_Model): void
+saveGame(void): void
-resetDecks(void): void
+ getGameWinningPlayer(void): Player
+topCard(void): Card
+hasActedThisTurn(void): boolean
+executeAITurn(void): void
-resetEventState(void): void
StateSnapShot
-previousHand: GameEvent
-currentPlayerEvent: int
-playerState: List<Player>
-playDeckState: Deck
-playPileState: Stack<Card>
-direction: Direction
-skipCount: int
-hasActedThisTurn: boolean
-WildColorChoice: CardColor
-waitingForColorChoice: boolean
+serialVersionUID: long = 1L
-isLightSide: boolean
+StateSnapShot(GameEvent: previousHand, int: currentPlayer, List<Player>: playerState, Deck: playDeckState, Stack<Card>: playPileState, Direction: direction, int: skipCount, boolean: hasActedThisTurn, CarcColor: WildColorChoice, boolean: waitingForColorChoice, boolean: isLightSide)
+getCurrentPlayerIndex(void): int
+getPlayersState(void): List<Player>
+getPlayDeckState(void): Deck
+getPlayPileState(void): Stack<Card>
+getDirection(void): Direction
+getSkipCount(void): int
+getHasActedThisTurn(void): boolean
+getWildColorChoice(void): CardColor
+getWaitingForColorSelection(): boolean
+getIsLightSide(): boolean
+getPreviousHand(void): GameEvent
<<interface>>
AIStrategy
+chooseCard(Player: player, Card: topCard, UNO_Model: game): int
+chooseWildColor(Player: aiPlayer, boolean: isLightSide): CardColor
+getDelayMilliseconds(void): int
+setDelayMilliseconds(int: delayMilliseconds): void
+serialVersionUID: long = 1L
+serialVersionUID: long = 1L
BasicAIStrategy
-random: Random
-delay_ms: int
+BasicAIStrategy(int: delay_ms)
+chooseCard(Player: player, Card: topCard, UNO_Model: game): int
+chooseWildColor(Player: aiPlayer, boolean: isLightSide): CardColor
+getDelayMilliseconds(void): int
+setDelayMilliseconds(int: delayMilliseconds): void
Use
CardComponent
-card: Card
-cardIndex: int
-isPlayable: boolean
-useButton: JButton
-cardImage: BufferedImage
+CardComponent(Card: card, int: index, UNO_CONTROLLER: controller)
-initializeComponents(UNO_CONTROLLER: controller): void
+getUseButton(void): JButton
+getCard(void): Card
+isPlayable(void): boolean
-getImagePath(void): String
-setupLayout(void): void
-drawCard(Graphics: g): void
+getCardIndex(void): int
+setPlayable(boolean: playable): void
-updateBorder(void): void
-loadCardImage(void): void
Player
-name: String
-hand: ArrayList<Card>
-score: int
-serialVersionUID: long = 1L
+Player(String: name)
+getName(void): String
+drawCard(Card: drawnCard): void
+playCard(int: cardNumber): Card
+removeCard(int: cardNumber): void
+printHand(void): void
+takeTurn(Scanner: input): int
+clearHand(void): void
+getHand(void): ArrayList<Card>
+handSize(void): int
+addScore(int: points): void
+getScore(void): int
<<interface>>
UNO_View
+handleGameEvent(GameEvent event): void
+getDrawButton(void): JButton
+getNextPlayerButton(void): JButton
<<enumeration>>
CardColor
RED
BLUE
GREEN
YELLOW
WILD
PURPLE
TEAL
ORANGE
PINK
+isLight(void): boolean
+isDark(void): boolean
+getDarkCounterpart(void): CardColor
+getLightCounterpart(void): CardColor
<<enumeration>>
CardType
ZERO
ONE
TWO
THREE
FOUR
FIVE
SIX
SEVEN
EIGHT
NINE
DRAW_ONE
DRAW_FIVE
REVERSE
SKIP
SKIP_EVERYONE
WILD
WILD_DRAW_TWO
WILD_DRAW_COLOR
FLIP
+isNumberCard(): boolean
+isActionCard(): boolean
+isWildCard(): boolean
+getPointValue(): int
Use
Deck
-deck: ArrayList<Card>
-serialVersionUID: long = 1L
+Deck()
-initializeDeck(void): void
+getDeck(void): ArrayList<Card>
+drawCard(void): Card
+shuffle(void): void
+setDeck(ArrayList<Card>: deck): void
<<enumeration>>
Direction
CLOCKWISE
CONTERCLOCKWISE
+toString(void): String
Use
Card
#isLightSideActive: boolean
#lightColor: Color
#darkColor: Color
#lightType: CardType
#darkType: CardType
-serialVersionUID: long = 1L
+action(UNO_GAME: model, Player:player): boolean
+playableOnTop(Card: otherCard): boolean
+getType(void): CardType
+getColor(void): Color
+flip(void): void
+getActiveSide(void): boolean
+printCard(void): void
Extends
Extends
Extends
Extends
Extends
Extends
Extends
GameEvent
-card: Card
-winningPlayer: Player
-type: EventType
-direction: Direction
-currentPlayer: Player
-enableNextPlayer: boolean
-message: String
-WildColorChoice: CardColor
-data: Map<String, Object>
-enableDrawButton: boolean
-serialVersionUID: long = 1L
+GameEvent(EventType: type, Player: currentPlayer, Player: winningPlayer, Card: card, String: message, Direction: direction, boolean: enableNextPlayer, boolean: enableDrawButton, CardColor: wildColorChoice)
+getBooleanData(String: key): boolean
+isEnableDrawButton(void): boolean
+getDirection(void): Direction
+getType(void): EventType
+setData(String: key, Object: value
+getCard(void): Card
+getMessage(void): String
+getCurrentPlayer(void): Player
+getWildColorChoice(void): CardColor
+isEnableNextPlayer(void): boolean
+getWinningPlayer(void): Player
+getData(String: key): Object
<<enum>> EventType
GAME_STATE_CHANGED
CARD_PLAYED
CARD_DRAWN
ROUND_WON
GAME_WON
AI_THINKING
MESSAGE
COLOR_SELECTION_NEEDED
COLOR_SELCTION_COMPLETE
PLAYER_CHANGED
DIRECTION_FLIPPED
SCORES_UPDATED
WildCard
-color_input: Scanner
+WildCard()
+playableOnTop(Card: otherCard): boolean
+action(UNO_Game: model, Player: player): boolean
+applyChosenColor(CardColor: chosenColor, boolean: isLightSide): void
DrawXCard
+DrawXCard(Color: light_color, Color: dark_color)
+action(UNO_Game: model, Player: player): boolean
+playableOnTop(Card: otherCard): boolean
SkipCard
+SkipCard(Color: light_color, Color: dark_color)
+action(UNO_Game: model, Player: player): boolean
+playableOnTop(Card: otherCard): boolean
WildDrawCard
+WildDrawCard()
+action(UNO_Game: model, Player: player): boolean
+playableOnTop(Card: otherCard): boolean
+executeDrawAction(CardColor: chosenColor, boolean: isLightSide, UNO_Game: model, Player: player): void
ReverseCard
+ReverseCard(Color: light_color, Color: dark_color)
+action(UNO_Game: model, Player: player): boolean
+playableOnTop(Card: otherCard): boolean
NumberCard
+NumberCard(Color: light_color, Color: dark_color, CardType: number)
+action(UNO_Game: model, Player: player): boolean
+playableOnTop(Card: otherCard): boolean

Sequence Diagram:

Sequence Diagram:

## Data structures:

**Changes made to UML and data Structures from Milestone 3:**
Milestone 4 introduced some structural updates to support undo/redo, saving/loading the game, and game restart functionality

1. Added Undo/Redo System:
   The undo/redo system allows the game to revert or re-apply previous states for full turnlevel history control.
    a. Introduced two main data structures in the model:
        i. private Stack<StateSnapShot> undoStack; stores previous game states so the model can revert one step backward.
        ii. private Stack<StateSnapShot> redoStack; stores states that were undone so they can be re-applied.
    b. UML updated to show the model maintaining associations to multiple StateSnapShot object

2. New StateSnapShot class:
   This class captures a copy of the entire game state so it can be restored exactly later.
    a. Stores copies of players, deck, play pile, direction, and event state. This allows complete reconstruction of the game at any moment.
    b. Appears in UML as new Component connected to the model

3. Model Modifications
   The Model was expanded to manage saving/loading state, restoring snapshots, and handling turn history.
    a. Added methods : captureState(), restoreState(), undo(), redo(), saveGame(), loadGame().
    b. Model now drives state restoration logic and supports game restarting

4. Controller Modifications
   The Controller was extended to forward new UI actions (undo/redo/load/save) into model operations.
    a. Added event handling for Undo, Redo, Save Game, Load Game, and New Game buttons, connecting UI actions to model logic
    b. The controller now delegates these features to the Model.

5. Frame Modifications:
   The Frame was updated to visually support the new functionality and show the game's interaction controls.
    a. Added new UI components: Undo / Redo / Save / Load / Restart buttons
    b. Updated layout to display game status and display the new buttons accordingly
    c. UML updated to include the new UI controls.

6. Serialization Support Across Classes
   Serialization enables saving and restoring the complete game state in external files.
   a. Model, Snapshot, and any related classes now implement Serializable to allow full game persistence.
   b. UML updated with the «serializable» stereotype to document all classes supporting file-based state storage.