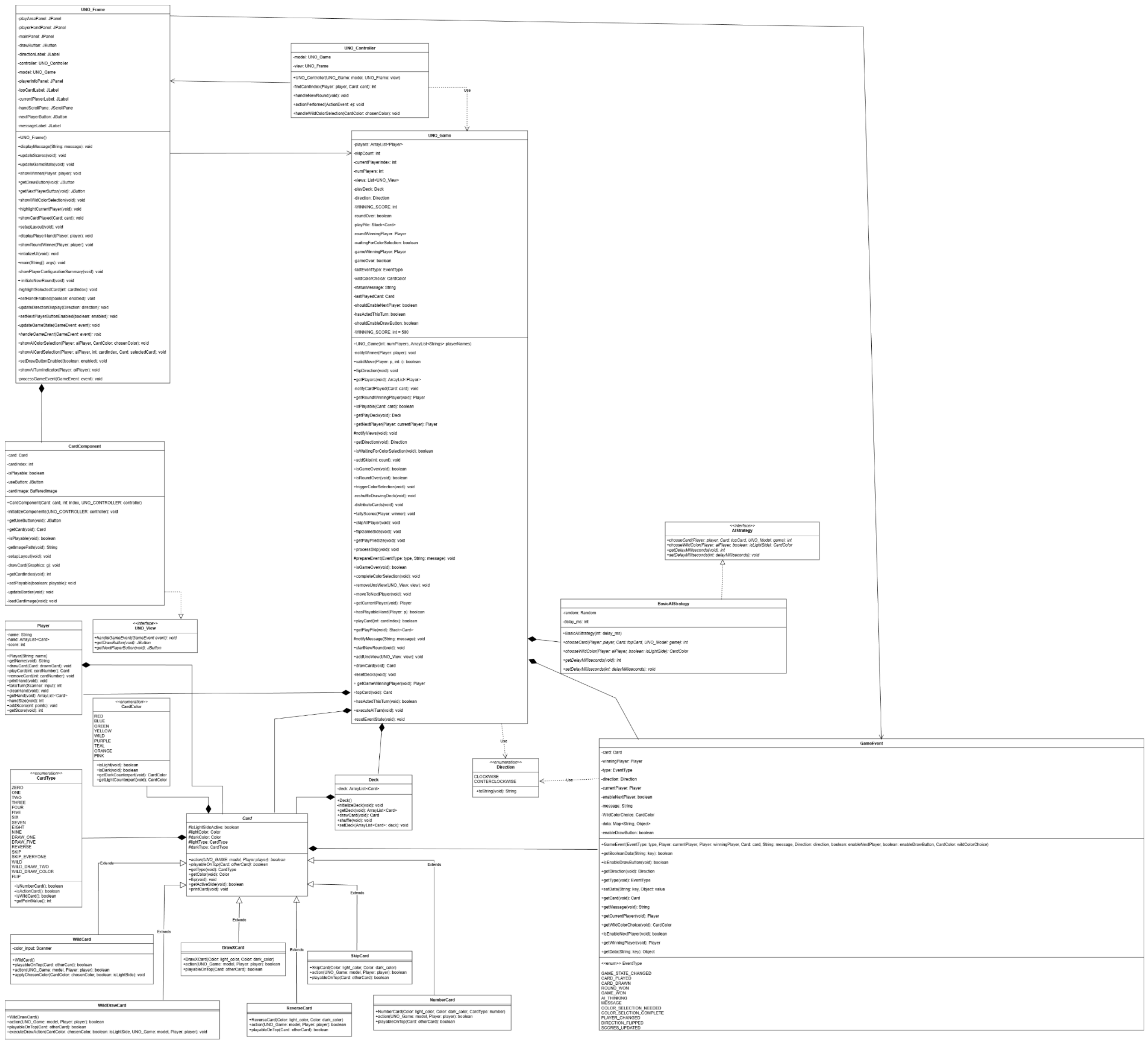
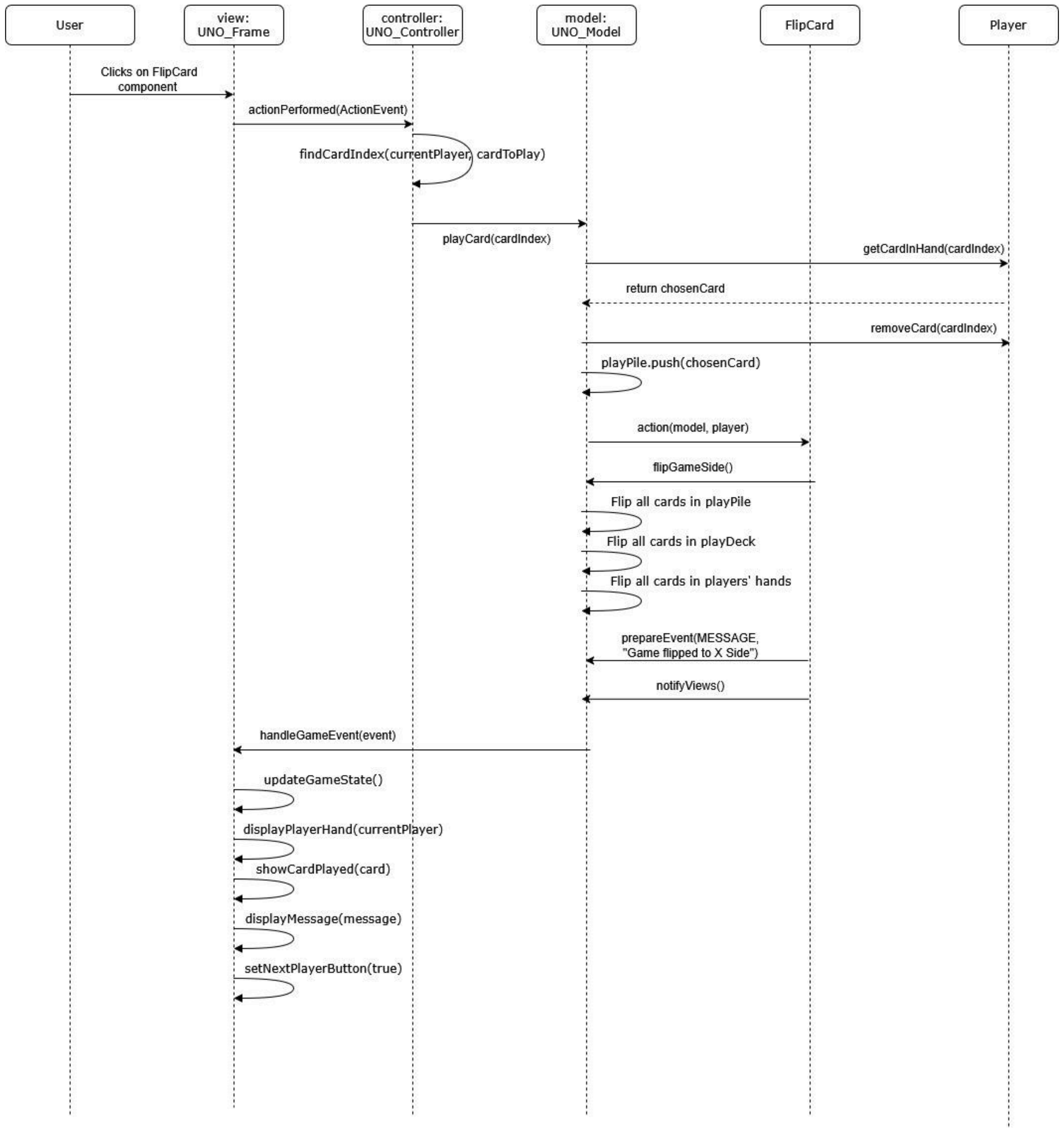
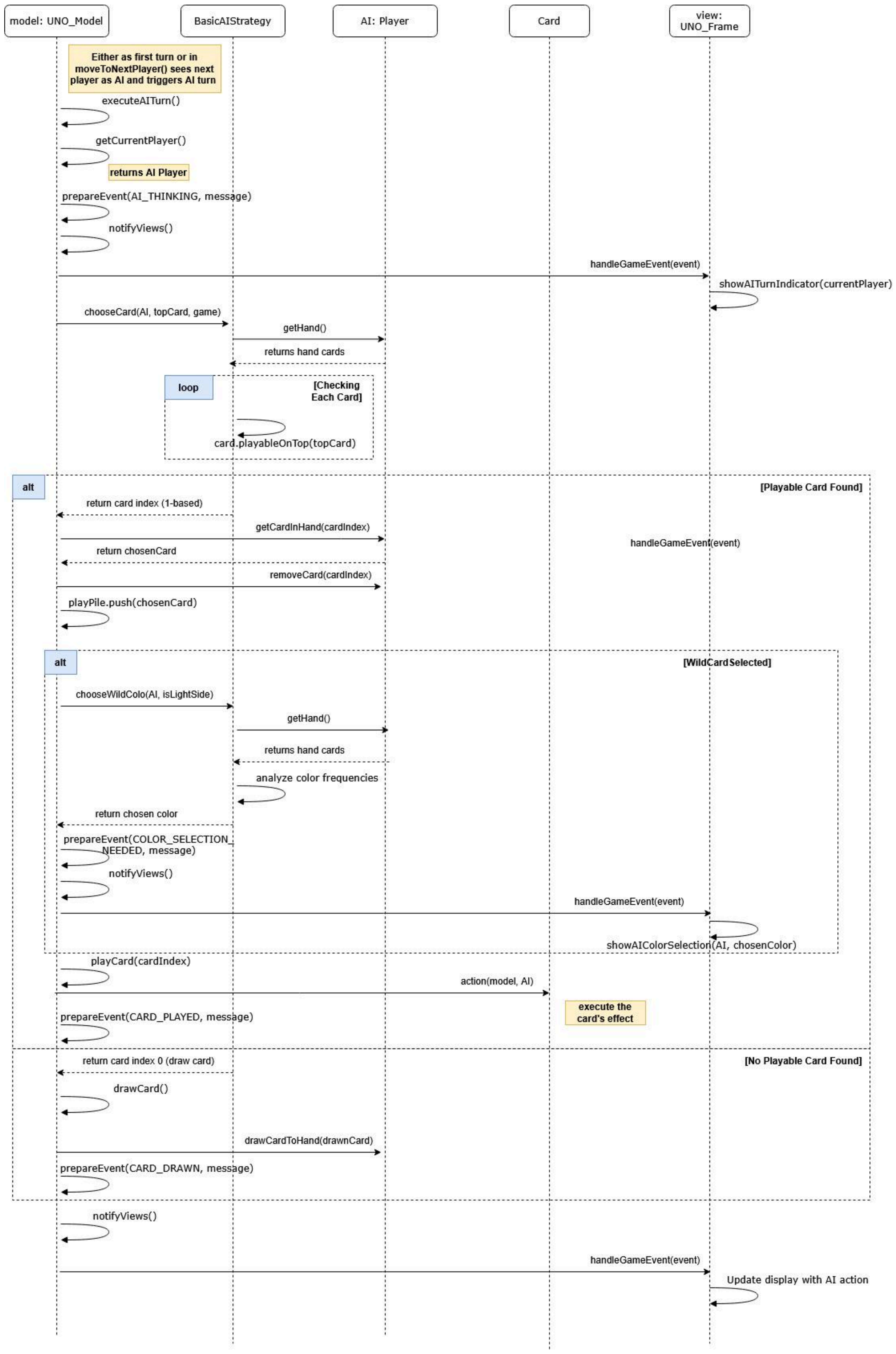
Uml Diagram:


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
-roundOver: boolean
-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
+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
+addUnoView(UNO_View: view): void
+drawCard(void): Card
-resetDecks(void): void
+ getGameWinningPlayer(void): Player
+topCard(void): Card
+hasActedThisTurn(void): boolean
+executeAITurn(void): void
-resetEventState(void): void
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
<<Interface>>
AIStrategy
+chooseCard(Player: player, Card: topCard, UNO_Model: game): int
+chooseWildColor(Player: aiPlayer, boolean: isLightSide): CardColor
+getDelayMilliseconds(void): int
+setDelayMilliseconds(int: delayMilliseconds): void
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
Player
-name: String
-hand: ArrayList<Card>
-score: int
+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
Deck
-deck: ArrayList<Card>
+Deck()
-initializeDeck(void): void
+getDeck(void): ArrayList<Card>
+drawCard(void): Card
+shuffle(void): void
+setDeck(ArrayList<Card>: deck): void
Use
<<enumeration>>
Direction
CLOCKWISE
CONTERCLOCKWISE
+toString(void): String
Use
Card
#isLightSideActive: boolean
#lightColor: Color
#darkColor: Color
#lightType: CardType
#darkType: CardType
+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
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
GameEvent
-card: Card
-winningPlayer: Player
-type: EventType
-direction: Direction
-currentPlayer: Player
-enableNextPlayer: boolean
-message: String
-WildColorChoice: CardColor
-data: Map<String, Object>
-enableDrawButton: boolean
+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

Sequence Diagram: Flipping to dark side/light side


User
view: UNO_Frame
controller: UNO_Controller
model: UNO_Model
FlipCard
Player
Clicks on FlipCard component
actionPerformed(ActionEvent)
findCardIndex(currentPlayer, cardToPlay)
playCard(cardIndex)
getCardInHand(cardIndex)
return chosenCard
removeCard(cardIndex)
playPile.push(chosenCard)
action(model, player)
flipGameSide()
Flip all cards in playPile
Flip all cards in playDeck
Flip all cards in players' hands
prepareEvent(MESSAGE, "Game flipped to X Side")
notifyViews()
handleGameEvent(event)
updateGameState()
displayPlayerHand(currentPlayer)
showCardPlayed(card)
displayMessage(message)
setNextPlayerButton(true)

Sequence Diagram: Ai bot taking a turn


model: UNO_Model
BasicAIStrategy
AI: Player
Card
view: UNO_Frame
Either as first turn or in moveToNextPlayer() sees next player as AI and triggers AI turn
executeAITurn()
getCurrentPlayer()
returns AI Player
prepareEvent(AI_THINKING, message)
notifyViews()
handleGameEvent(event)
showAITurnIndicator(currentPlayer)
chooseCard(AI, topCard, game)
getHand()
returns hand cards
loop
[Checking Each Card]
card.playableOnTop(topCard)
alt
[Playable Card Found]
return card index (1-based)
getCardInHand(cardIndex)
handleGameEvent(event)
return chosenCard
removeCard(cardIndex)
playPile.push(chosenCard)
alt
[WildCardSelected]
chooseWildColo(AI, isLightSide)
getHand()
returns hand cards
analyze color frequencies
return chosen color
prepareEvent(COLOR_SELECTION_NEEDED, message)
notifyViews()
handleGameEvent(event)
showAIColorSelection(AI, chosenColor)
playCard(cardIndex)
action(model, AI)
execute the card's effect
prepareEvent(CARD_PLAYED, message)
[No Playable Card Found]
return card index 0 (draw card)
drawCard()
drawCardToHand(drawnCard)
prepareEvent(CARD_DRAWN, message)
notifyViews()
handleGameEvent(event)
Update display with AI action

Data structures:

ArrayList:

Used in:
-Deck (private ArrayList<Card> deck;)
-UNO_GAME(private ArrayList<Card> drawPile;)
-Player (private ArrayList<Card> hand;)

Why:
-Has a dynamic size
-Can store objects
-able to easily access indexes
-Able to easily remove objects

Enum:

Used in:
-Color (RED, BLUE, YELLOW, etc.)
-CardType (DRAW_ONE, WILD, etc.)
-Direction (CLOCKWISE or COUNTERCLOCKWISE)

Why:
-It’s a fixed set of constants
-Prevents invalid values
-Has built in helper functions

Stack:

Used in:
-UNO_GAME (private Stack<Card> playPile;)

Why:
-The Uno play pile is LIFO since the most recently played card is at the top

Changes made to UML and data structures from Milestone 1:

The project now follows the MVC (Model-View-Controller) architecture. The updated UML diagram includes the MVC-based implementation of UNO_Game, UNO_View (and UNO_Frame), and UNO_Controller, along with the newly introduced Card Component class. Several methods from Milestone 1 have been revised and reorganized to align with this structure. The core game logic has been transferred from UNO_Game to the Model, establishing clear relationships among the model, view, and controller to create a more modular, maintainable, and scalable system.

Changes made to UML and data structures from Milestone 1:

The project now follows the MVC (Model-View-Controller) architecture. The updated UML diagram includes the MVC-based implementation of UNO_Game, UNO_View (and UNO_Frame), and UNO_Controller, along with the newly introduced Card Component class. Several methods from Milestone 1 have been revised and reorganized to align with this structure. The core game logic has been transferred from UNO_Game to the Model, establishing clear relationships among the model, view, and controller to create a more modular, maintainable, and scalable system.

Changes made to UML and data structures from Milestone 2:

The controller now only reacts to user actions and forwards them to the model. It no longer updates the view directly and no longer touches buttons, labels, or layout. It simply calls model methods and lets the model drive the flow of the game. This makes the controller simpler and focused on input instead of preparation.

For the model, it no longer knows anything about the view. Instead it sends the same event containing the facts of what happened, such as which card was played or whether a color must be chosen. The view reads the event and decides how to update the interface. The model never draws anything, and the view never runs games rules. Only the event connects them, giving each part a clean and independent role.

Explanation of AI Player Strategy:
All AI strategy implements the AIStrategy interface for methods for choosing a card and choosing a colour for wild cards, which enables for multiple strategies to be developed in future revisions. Milestone 3 utilizes a basic AI strategy that employs a straightforward decision making of choosing the first valid card (from left to right) from in the player's hand to play.

I. Card Selection Strategy

i. Sequential Hand Scanning

- The AI scans its hand from first to last card (index 0 to hand size-1)
- It plays the **first legally playable card** it encounters that matches the current top card
- This creates predictable but consistent gameplay behavior

ii. Simple Playability Check

- For each card in hand, the AI calls card.playableOnTop(topCard)
- This method checks if the card matches the top card by either:

    - Same color
    - Same type/number
    - Or if it's a Wild card (always playable)

iii. Draw Decision

- If no playable cards are found after scanning the entire hand, the AI chooses to draw a card (returns 0)
- The drawn card is automatically evaluated for playability in the next turn

<u>II. Wild Card Color Selection</u>

i. Color Frequency Analysis

- The AI counts how many cards of each color it holds in its hand
- It excludes Wild cards from this count since they don't contribute to colour strategy\

ii. Majority

- The AI chooses the color that appears most frequently in its hand
- This maximizes the probability of having follow-up plays in subsequent turns