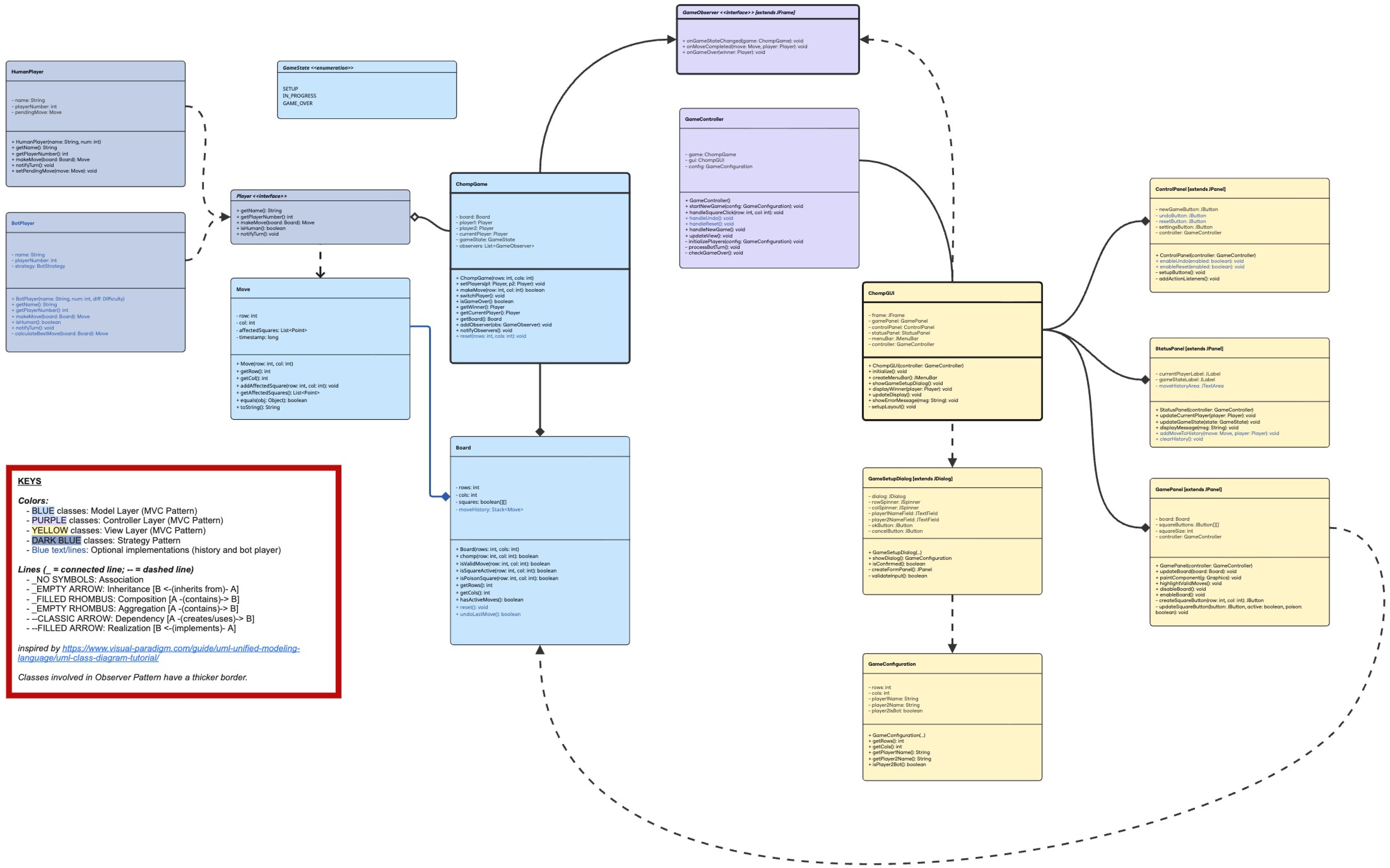


```
HumanPlayer
- name: String
- playerNumber: int
- pendingMove: Move
+ HumanPlayer(name: String, num: int)
+ getName(): String
+ getPlayerNumber(): int
+ makeMove(board: Board): Move
+ notifyTurn(): void
+ setPendingMove(move: Move): void

BotPlayer
- name: String
- playerNumber: int
- strategy: BotStrategy
+ BotPlayer(name: String, num: int, diff: Difficulty)
+ getName(): String
+ getPlayerNumber(): int
+ makeMove(board: Board): Move
+ isHuman(): boolean
+ notifyTurn(): void
- calculateBestMove(board: Board): Move

GameState <<enumeration>>
SETUP
IN_PROGRESS
GAME_OVER

Player <<interface>>
+ getName(): String
+ getPlayerNumber(): int
+ makeMove(board: Board): Move
+ isHuman(): boolean
+ notifyTurn(): void

Move
- row: int
- col: int
- affectedSquares: List<Point>
- timestamp: long
+ Move(row: int, col: int)
+ getRow(): int
+ getCol(): int
+ addAffectedSquare(row: int, col: int): void
+ getAffectedSquares(): List<Point>
+ equals(obj: Object): boolean
+ toString(): String

ChompGame
- board: Board
- player1: Player
- player2: Player
- currentPlayer: Player
- gameState: GameState
- observers: List<GameObserver>
+ ChompGame(rows: int, cols: int)
+ setPlayers(p1: Player, p2: Player): void
+ makeMove(row: int, col: int): boolean
+ switchPlayer(): void
+ isGameOver(): boolean
+ getWinner(): Player
+ getCurrentPlayer(): Player
+ getBoard(): Board
+ addObserver(obs: GameObserver): void
+ notifyObservers(): void
+ reset(rows: int, cols: int): void

Board
- rows: int
- cols: int
- squares: boolean[][]
- moveHistory: Stack<Move>
+ Board(rows: int, cols: int)
+ chomp(row: int, col: int): boolean
+ isValidMove(row: int, col: int): boolean
+ isSquareActive(row: int, col: int): boolean
+ isPoisonSquare(row: int, col: int): boolean
+ getRows(): int
+ getCols(): int
+ hasActiveMoves(): boolean
+ reset(): void
+ undoLastMove(): boolean

GameObserver <<interface>> [extends JFrame]
+ onGameStateChanged(game: ChompGame): void
+ onMoveCompleted(move: Move, player: Player): void
+ onGameOver(winner: Player): void

GameController
- game: ChompGame
- gui: ChompGUI
- config: GameConfiguration
+ GameController()
+ startNewGame(config: GameConfiguration): void
+ handleSquareClick(row: int, col: int): void
+ handleUndo(): void
+ handleReset(): void
+ handleNewGame(): void
+ updateView(): void
- initializePlayers(config: GameConfiguration): void
- processBotTurn(): void
- checkGameOver(): void

ChompGUI
- frame: JFrame
- gamePanel: GamePanel
- controlPanel: ControlPanel
- statusPanel: StatusPanel
- menuBar: JMenuBar
- controller: GameController
+ ChompGUI(controller: GameController)
+ initialize(): void
+ createMenuBar(): JMenuBar
+ showGameSetupDialog(): void
+ displayWinner(player: Player): void
+ updateDisplay(): void
+ showErrorMessage(msg: String): void
- setupLayout(): void

GameSetupDialog [extends JDialog]
- dialog: JDialog
- rowSpinner: JSpinner
- colSpinner: JSpinner
- player1NameField: JTextField
- player2NameField: JTextField
- okButton: JButton
- cancelButton: JButton
+ GameSetupDialog(...)
+ showDialog(): GameConfiguration
+ isConfirmed(): boolean
- createFormPanel(): JPanel
- validateInput(): boolean

GameConfiguration
- rows: int
- cols: int
- player1Name: String
- player2Name: String
- player2IsBot: boolean
+ GameConfiguration(...)
+ getRows(): int
+ getCols(): int
+ getPlayer1Name(): String
+ getPlayer2Name(): String
+ isPlayer2Bot(): boolean

ControlPanel [extends JPanel]
- newGameButton: JButton
- undoButton: JButton
- resetButton: JButton
- settingsButton: JButton
- controller: GameController
+ ControlPanel(controller: GameController)
+ enableUndo(enabled: boolean): void
+ enableReset(enabled: boolean): void
- setupButtons(): void
- addActionListeners(): void

StatusPanel [extends JPanel]
- currentPlayerLabel: JLabel
- gameStateLabel: JLabel
- moveHistoryArea: JTextArea
+ StatusPanel(controller: GameController)
+ updateCurrentPlayer(player: Player): void
+ updateGameState(state: GameState): void
+ displayMessage(msg: String): void
+ addMoveToHistory(move: Move, player: Player): void
+ clearHistory(): void

GamePanel [extends JPanel]
- board: Board
- squareButtons: JButton[][]
- squareSize: int
- controller: GameController
+ GamePanel(controller: GameController)
+ updateBoard(board: Board): void
+ paintComponent(g: Graphics): void
+ highlightValidMoves(): void
+ disableBoard(): void
+ enableBoard(): void
- createSquareButton(row: int, col: int): JButton
- updateSquareButton(button: JButton, active: boolean, poison: boolean): void
```

KEYS

Colors:
- BLUE classes: Model Layer (MVC Pattern)
- PURPLE classes: Controller Layer (MVC Pattern)
- YELLOW classes: View Layer (MVC Pattern)
- DARK BLUE classes: Strategy Pattern
- Blue text/lines: Optional implementations (history and bot player)

Lines (_ = connected line; -- = dashed line)
- _NO SYMBOLS: Association
- _EMPTY ARROW: Inheritance [B <-(inherits from)- A]
- _FILLED RHOMBUS: Composition [A -(contains)-> B]
- _EMPTY RHOMBUS: Aggregation [A -(contains)-> B]
- --CLASSIC ARROW: Dependency [A -(creates/uses)-> B]
- --FILLED ARROW: Realization [B <-(implements)- A]

inspired by https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

Classes involved in Observer Pattern have a thicker border.

UML Class Diagram

- Above is the UML Class Diagram for the Game Chomp used for Assignment 3 PDP Fall 2025
- Use this Miro Board to view a less blurry version of the above diagram

Logistics

Game Flow

- ChompGUI displays GameSetupDialog to get configuration
- GameController initializes ChompGame with Board and Players
- For HumanPlayer turns: enable board, wait for square click
- For BotPlayer turns: disable board, calculate and execute move automatically
- After each move: update board, switch players, check for game over
- Notify all observers to refresh GUI components

Program Loop and Rendering

- This program does not use a traditional game loop. It uses Swing's event dispatch thread.
- Implement *GameObserver* pattern to automatically update views when model changes
- ChompGUI coordinates overall rendering
- ActionListener for buttons
- Call `repaint()`:
  - After a move is made
  - After board reset
  - When switching between players
  - When game ends
- Call `revalidate()`:
  - After dynamically adding/removing components
  - After changing component sizes
  - After updating text in labels/text areas
  - After changing layout

Required Imports

Model Classes:

```
java.util.List
java.util.ArrayList
java.util.Stack
java.awt.Point
```

View Classes:

```
javax.swing.*
java.awt.*
java.awt.event.*
```

Controller Class:

```
javax.swing.SwingUtilities
```

Class Purpose Guidance

Model Layer

- ChompGame: Central coordinator for game logic and state management; note that this is event-driven and does not have a traditional while game loop
    - Init board w specified dimensions
    - Assign 2 players, toggle between them, identify whose move is happening
    - Move (through Board Class)
    - Check and update game state
    - Register and notify observers
- Board: Manages the chocolate bar grid and move validation
    - Check whether move is legal
    - Executes chomp (set square and all squares right and above to False)
    - Check whether chomped poison square
    - Any other fundamental gameplay logic (history?)
- Move: Object representing move action (good for history implementation)
    - Compute and return affected squares
    - Return move in String
- GameState [enumeration]: quite straightforward, notified by ChompGame to update ChompGUI

Strategy Layer

- Player [interface]: Defines contract for all player types
    - Player name
    - Player number (1 or 2)
    - Determines human or bot player
    - Notify turn
    - Determines and returns next move

Controller Layer

- GameObserver: Observer notified by ChompGame to update ChompGUI
    - (See ChompGUI class for concrete implementation)
    - Observed events (game state changes, completed move, game over)
- GameController: Mediates between View and Model, handles all user interactions
    - Start new game with ChompGame functionalities and init ChompGUI
    - Handle when user click a square and update GUI accordingly
    - Process bot turn (optional)
    - Other functionalities' Model -> View

View Layer

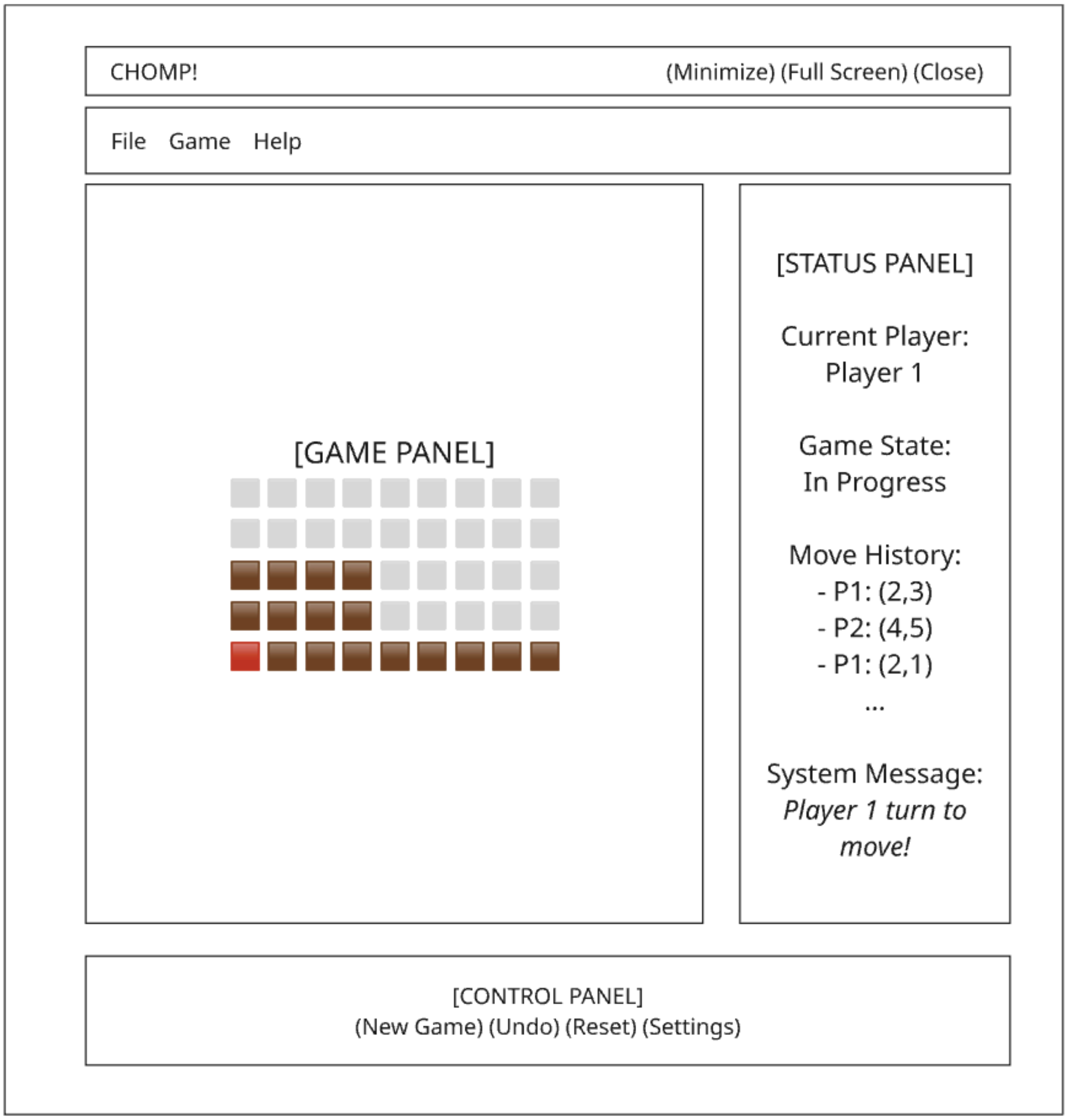
- ChompGUI: Main application window that coordinates all GUI components
    - Sets up the frame, adds components, sets layout

    - Create menu bar
    - Displays modal dialog for game configuration + errors
    - Update display with repaint() or revalidates()
    - Displays winner
    - GameObserver implementations:
        - Update all panels on game state changed
        - Update status and board when a move is completed
        - Display winner and disable board on game over

- StatusPanel: Displays current player, game state, and move history
    - Update current player text
    - Update game state text
    - Changes history scrollable list based on player's action
    - Show any in-game messages
- GamePanel: Visual representation of in-game elements such as the chocolate bar with clickable squares
    - Calls repaint() after board updates
    - Fill squares with colors (indicator of status: active, chomped, poison)
    - Enable/disable move buttons
    - Any custom appearance update relating to game board
- ControlPanel: Provides buttons for game control actions
    - Start new game
    - Setting dialogue
    - Undo/reset buttons
    - Any custom appearance update relating to game control actions
- GameSetupDialog: Modal dialog for configuring a new game
    - Display dialogues
    - Form components handling (ok/cancel)
- GameConfiguration: Game setup parameters
    - Getter for all game parameters (player names, bot status)

GUI Description

- See Wireframe below
- Square color-coding example:
    - brown: active
    - white: chomped
    - red: poison



CHOMP! (Minimize) (Full Screen) (Close)

File Game Help

[GAME PANEL]

[STATUS PANEL]

Current Player:
Player 1

Game State:
In Progress

Move History:
- P1: (2,3)
- P2: (4,5)
- P1: (2,1)

...

System Message:
Player 1 turn to move!

[CONTROL PANEL]
(New Game) (Undo) (Reset) (Settings)