Chomp Game - Design Notes

Design Patterns

1. Model-View-Controller (MVC)

- **Model**: `Board` and `Cell` classes manage game state
- **View**: `Display`, `Frame`, `GamePanel`, and `InfoPanel` handle all GUI rendering
- **Controller**: `CHOMPGame` coordinates between model and view, handles user input

2. Template Method Pattern

- `Game` abstract class defines the skeleton for turn-based games
- `CHOMPGame` provides concrete implementation of abstract methods

3. Event-Driven Architecture

- `CHOMPGame` implements `ActionListener` to respond to button clicks
- Decouples user input from game logic

Class Descriptions

Driver

Entry point for the application. Creates a `CHOMPGame` instance with initial parameters (board dimensions, player names) and starts the game.

Game (Abstract)

Base class for turn-based games. Manages player array and current player tracking. Subclasses must implement `playTurn()`, `isGameOver()`, and `getWinner()`. Provides `switchPlayer()` utility.

CHOMPGame

Main game controller. Orchestrates the game loop by coordinating `Board` updates and `Display` refreshes. Implements `ActionListener` to handle button clicks from `GamePanel`. When a move is made, updates the board, checks for game-over conditions, switches players, and refreshes the display.

Board

Manages the 2D grid of `Cell` objects. `initialize()` creates all cells as uneaten. `makeMove(row, col)` implements Chomp logic: marks the selected cell and all cells to the right and above as eaten. Bottom-left cell (0,0) is the "poison" square.

Cell

Represents a single chocolate square. Tracks whether it's been eaten via boolean flag. Simple `eat()` method to mark as consumed.

Player

Stores player information (name). Used to track whose turn it is and determine the winner.

Display

View manager that bridges game logic and GUI components. Creates and manages `Frame`, `GamePanel`, and `InfoPanel`. `showBoard()` refreshes the visual representation. `setActionListener()` connects `CHOMPGame` as the event handler for all buttons.

Frame

Main application window extending `JFrame`. Contains `GamePanel` (center) and `InfoPanel` (top/bottom). `setUpFrame()` configures window properties and layout manager.

GamePanel

Extends `JPanel` and contains a 2D array of `JButton` objects matching board dimensions. Each button represents a cell. When clicked, buttons trigger `actionPerformed()` in `CHOMPGame`. Override `paintComponent()` to update button states (enabled/disabled, color) based on `Board` state.

InfoPanel

Extends `JPanel` and displays game status via `JLabel`. Shows current player's turn or game-over message. `updateInfo(String)` refreshes the label text.

More Implementation Notes

Game Loop Management

`CHOMPGame` manages the program loop. After each move: call `board.makeMove()`, check `isGameOver()`, call `display.showBoard()` to trigger `repaint()`, then `switchPlayer()`.

Event Handling

- Each button in `GamePanel` should have its row/column stored (use `ActionCommand` or button properties). You could also use a custom `Button` subclass if that helps.
- `CHOMPGame.actionPerformed()` extracts coordinates from the event source
- Parse coordinates and validate with `board.isCellAvailable()`

GUI Layout

- Use `BorderLayout` for `Frame`: `InfoPanel` at TOP, `GamePanel` at CENTER
- Use `GridLayout` for `GamePanel` to arrange buttons in rows × cols grid
- Color scheme: Available cells (brown/tan), eaten cells (white/grey), poison cell (red/distinct color)

Game Logic

For `row, col` move:

- Loop through all cells where `r >= row` and `c >= col`
- Call `cell.eat()` on each to mark as eaten
- Ensure `isGameOver()` checks if the poison cell (0,0) is eaten. If
  - TRUE: current player loses, other player wins
  - FALSE: continue game, switch turns using `switchPlayer()`

UML Class Diagram

See Figure 1.

UI Wireframe

See Figure 2.

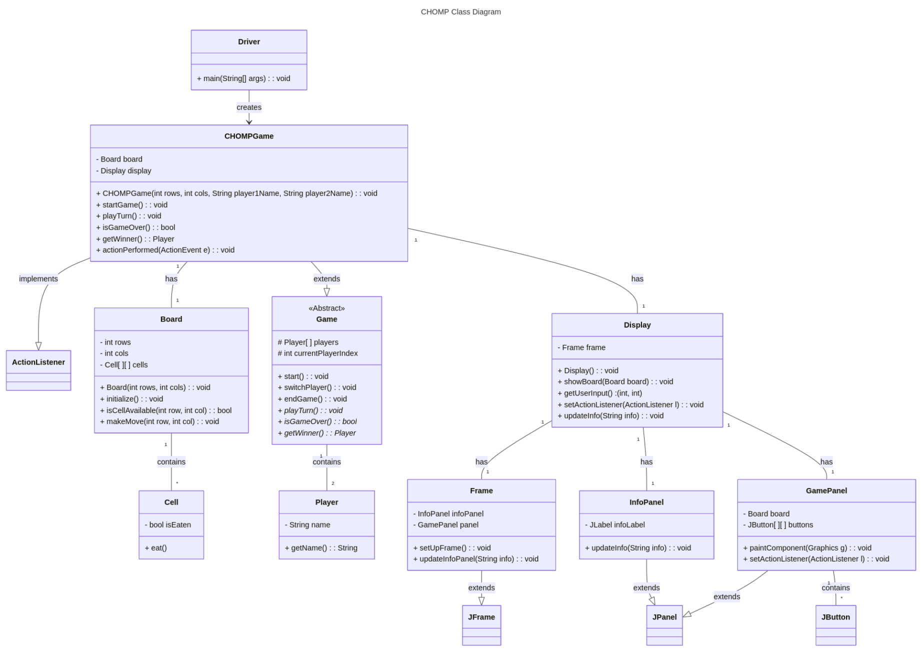

Figure 1: Chomp Game UML Diagram

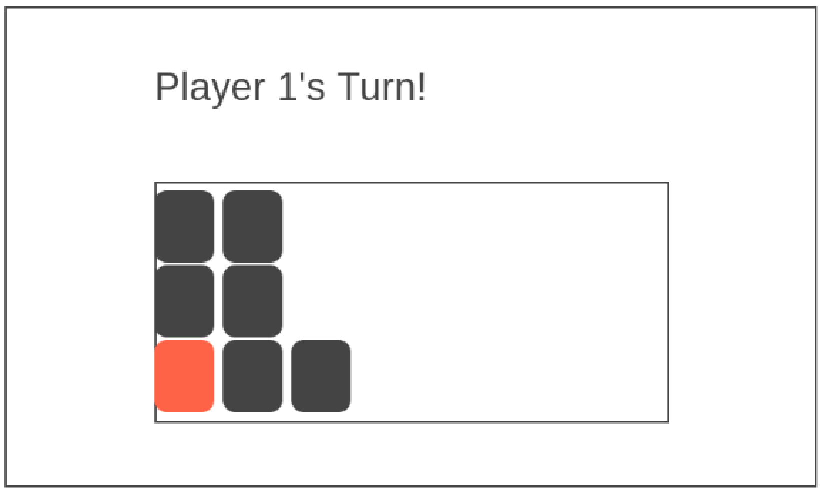

Figure 2: Wireframe