

**Class Notes**

**Main**

* **Purpose**: Acts as the entry point for the game. Its main responsibility is to initialize the GUI and start the game.
* **Methods**:
  + main(String[] args): Launches the application by creating an instance of ChompGameGUI. This method sets up the initial state of the game and begins the GUI.

**ChompGame**

* **Purpose**: Main controller for the game logic, managing the game state, player turns, and interactions with the board. It keeps track of the current player and handles game progression and end conditions.
* **Attributes**:
  + chocolateBar: Manages the grid of squares (each square is either eaten or uneaten).
  + players: Array holding the two players.
  + currentPlayerIndex: Tracks which player's turn it is.
  + gameState: Enum to represent the current state of the game (e.g., ongoing, win states).
* **Methods**:
  + ChompGame(int rows, int columns): Initializes the game with a specified grid size and sets up players.
  + initializeGame(): Sets up the game state by resetting the grid and preparing players for the start of a new game.
  + handleMove(int row, int col): Processes a player’s move, marking squares as eaten on the grid. It also checks for game-ending conditions and updates the game state as necessary.
  + switchPlayer(): Switches the turn to the other player after each move.
  + isGameOver(): Checks if the game has reached an end condition, specifically if only the "poison" square remains.
  + getGameState(): Returns the current game state for use in the GUI to determine if the game should continue or display an end result.
  + getCurrentPlayer(): Returns the current active player.
  + getChocolateBar(): Provides access to the ChocolateBar instance, allowing the GUI to retrieve the state of the grid.

**ChocolateBar**

* **Purpose**: Manages the state of the chocolate grid, marking squares as eaten or uneaten. This class is responsible for maintaining the grid layout and enabling updates based on player actions.
* **Attributes**:
  + grid: A 2D array representing the grid, where each element indicates if a square is uneaten (true) or eaten (false).
  + rows: Number of rows in the grid.
  + columns: Number of columns in the grid.
* **Methods**:
  + ChocolateBar(int rows, int columns): Sets up the grid with the specified dimensions, marking all squares as uneaten initially.
  + markSquaresAsEaten(int row, int col): Marks all squares to the right and below a selected square as eaten, indicating the result of a player’s move.
  + isSquareUneaten(int row, int col): Checks if a specific square on the grid is uneaten, used to determine whether a move is valid.
  + reset(): Resets the grid, marking all squares as uneaten. Called at the start of a new game.
  + getGrid(): Returns the grid state, allowing other classes (like ChocolateBarPanel) to access it for display.

**Player**

* **Purpose**: Represents a player in the game, tracking their name and turn status. Each player has a unique identity and takes turns during gameplay.
* **Attributes**:
  + name: The player’s name or identifier (e.g., "Player 1").
  + isActive: Indicates if it’s currently the player’s turn.
* **Methods**:
  + Player(String name): Constructor that initializes the player with a name.
  + setActive(boolean active): Sets the player’s active status, determining if it's their turn.
  + getName(): Returns the player’s name, used for display in the GUI.
  + isActive(): Checks if the player is active (i.e., if it’s their turn).

**GameState (Enum)**

* **Purpose**: Represents different game states to track the progress and conclusion of the game.
* **Values**:
  + ONGOING: The game is currently in progress.
  + PLAYER\_1\_WIN: Player 1 has won the game.
  + PLAYER\_2\_WIN: Player 2 has won the game.
  + GAME\_OVER: A generic end state indicating that the game has ended.

**ChompGameGUI**

* **Purpose**: Main window for the GUI, displaying the chocolate grid and handling player interactions. Manages the layout and interface, connecting the game logic to visual components.
* **Attributes**:
  + game: Instance of ChompGame, which manages the game state and player turns.
  + chocolateBarPanel: Custom JPanel (ChocolateBarPanel) that displays and updates the chocolate grid visually.
* **Methods**:
  + ChompGameGUI(int rows, int columns): Initializes the GUI window, sets up chocolateBarPanel, and defines the layout.
  + setupGUI(int rows, int columns): Adds chocolateBarPanel to the GUI, sets the initial layout, and adds listeners for handling user interactions.
  + updateDisplay(): Calls repaint() on chocolateBarPanel to refresh the grid based on the current state, showing eaten and uneaten squares.
  + checkGameState(): Checks the game state after each move, determining if a player has won or if the game should continue.

**ChocolateBarPanel (extends JPanel)**

* **Purpose**: Custom JPanel for rendering the chocolate grid, displaying the state of each square and updating it in real-time as moves are made.
* **Attributes**:
  + grid: 2D array that stores the state of each square, either uneaten (true) or eaten (false), reflecting the chocolate bar’s condition.
  + squareSize: Size of each square in pixels, allowing for flexible grid scaling.
* **Methods**:
  + ChocolateBarPanel(int rows, int columns): Sets up the grid based on the specified dimensions, marking all squares as uneaten at the start.
  + paintComponent(Graphics g): Overrides the JPanel method to custom-draw each square on the grid. Uneaten squares are drawn in one color, while eaten squares are drawn in another, providing a clear visual representation.
  + eatSquares(int row, int col): Updates the grid to mark all squares to the right and below the selected square as eaten. Calls repaint() to refresh the grid display in real-time.

**ButtonClickListener**

* **Purpose**: Handles click events on each square in the grid. When a square is clicked, it processes the move, updates the game state, and switches players if necessary.
* **Attributes**:
  + row: Row index of the button associated with this listener.
  + col: Column index of the button associated with this listener.
* **Methods**:
  + ButtonClickListener(int row, int col): Initializes the listener with the specific button’s row and column coordinates.
  + actionPerformed(ActionEvent e): Processes the click event, calling relevant methods in ChompGame to handle the move. This method updates the game state, marks squares as eaten, and switches to the next player.

**Chomp GUI documentation**

A screenshot of a yellow grid

Description automatically generatedA screenshot of a game

Description automatically generatedA screenshot of a game

Description automatically generatedA screenshot of a computer screen

Description automatically generated

The Chomp game interface consists of a straightforward layout with three main sections:

1. **Title and Player Turn Indicator** (Top of the Window)
   * Positioned at the top of the window, a label displays the current player's turn (e.g., "Player 1's Turn").
   * The title is set as "Chomp Game," displayed in the application window's title bar.
2. **Chocolate Grid** (Center of the Window)
   * The central portion of the window contains the chocolate grid, displayed as a panel (ChocolateBarPanel) that holds a grid of squares.
   * Each square represents a piece of chocolate, with colors distinguishing between uneaten and eaten squares.
3. **Reset Button** (Bottom of the Window)
   * At the bottom, a "Restart Game" button allows players to reset the game and start over. This button clears the board and resets the player turns.

**Component Descriptions**

1. **Player Turn Indicator**
   * **Component**: JLabel
   * **Purpose**: Displays which player's turn it is at any given time.
   * **Location**: Positioned at the top of the window.
   * **Style**: Displayed in a bold font for visibility, with a black color for clarity.
2. **Chocolate Grid Panel**
   * **Component**: ChocolateBarPanel (a subclass of JPanel)
   * **Purpose**: Displays the chocolate grid using a grid layout where each square’s color represents its state (uneaten or eaten).
   * **Colors**:
     + **Uneaten Squares**: Displayed in orange (Color.ORANGE) to indicate they are in play.
     + **Eaten Squares**: Displayed in dark gray (Color.DARK\_GRAY) to show they are no longer available.
     + **Poison Square**: The top-left square can optionally be displayed in red to indicate its significance as the losing square if selected.
   * **Borders**: Each square has a thin black border to visually separate them.
   * **Real-Time Updates**: The paintComponent method in ChocolateBarPanel is overridden to draw each square based on the game state, and repaint() is called after each move to update the display.
3. **Restart Button**
   * **Component**: JButton
   * **Purpose**: Provides a way to restart the game, resetting the board to its initial state.
   * **Location**: Positioned at the bottom of the window.
   * **Style**: A simple button with text "Restart Game."

**Look and Feel**

The application uses a clean and functional design, ensuring players can easily distinguish between the different states of each square on the grid.

* **Colors**:
  + **Uneaten Squares**: Orange color for squares that are still in play, providing high visibility.
  + **Eaten Squares**: Dark gray for squares that have been selected, making it easy to see unavailable sections of the grid.
* **Font and Styles**:
  + **Player Turn Indicator**: Displayed in a bold font, centered at the top for emphasis.
  + **Button Styling**: Standard button styling with padding for easy clicking.
* **Padding and Alignment**:
  + The chocolate grid panel is centered within the window, with padding around it to avoid a cluttered look.
  + The reset button is aligned at the bottom of the window, separated from the grid for clarity.

**Program Flow and Real-Time Updates**

* **Main Program Flow**:
  + The ChompGameGUI class is responsible for initializing and managing the main window, where the player interacts with the game.
  + The player’s actions are handled by a ClickListener, which captures clicks on the chocolate grid and updates the game state accordingly.
* **Real-Time Updates with paintComponent**:
  + **ChocolateBarPanel**: The paintComponent method in ChocolateBarPanel is overridden to draw each square in the chocolate grid. After each move, repaint() is called to ensure the grid displays the updated state.
  + **Game End Handling**: When the game reaches an end condition (the poison square is eaten), a message is displayed to announce the losing player, and players are given the option to restart the game.
* **Event Handling with ActionListener**:
  + The restart button is managed by an ActionListener in ChompGameGUI, which resets the game and clears the board when clicked.
  + Each click on a square in the chocolate grid triggers ClickListener to process the move, update the game state, and call repaint() to refresh the display.

**Sample Screenshots and Explanation**

Include the following screenshots in your documentation:

1. **Initial Game State**:
   * A screenshot showing the full grid with all squares in their uneaten state (orange).
   * Caption: “Initial Game State with all squares uneaten.”
2. **Mid-Game State**:
   * A screenshot showing the grid with some squares eaten (dark gray), representing the progress of the game.
   * Caption: “Mid-Game State with a mix of uneaten and eaten squares.”
3. **Game Over State**:
   * A screenshot displaying the game-over message indicating the losing player.
   * Caption: “Game Over State with a message indicating the losing player.”

Each screenshot should be labeled with a brief description of what it represents, so it’s clear how the GUI components come together to create the full game interface.

**Optional: Simple Wireframe Sketch Description**

If you prefer to include a wireframe sketch, here’s a guide to creating it:

1. **Title and Player Turn Indicator**: Place a label at the top of the window with "Player X's Turn."
2. **Chocolate Grid (ChocolateBarPanel)**: Draw a 5x5 grid representing the chocolate squares, with different colors for uneaten and eaten squares.
3. **Restart Button**: Add a button labeled "Restart Game" at the bottom.