**Comprehensive Documentation for Maze of Knowledge App**

This documentation covers every Java file used in the app. It explains what each file does, how its code is structured, and how you can modify aspects like time limits, movement speed, colors, fonts, and other UI properties.

**1. PreferencesManager.java**

**Purpose:**  
Manages user preferences (volume, theme, difficulty, etc.) using Java’s Preferences API. It centralizes configuration so that other parts of the app can retrieve and update settings easily.

**Code Explanation & Modification Tips:**

* **Initialization:**

java

CopyEdit

public static void initialize() {

prefs = Preferences.userNodeForPackage(VTUGamifiedQuizApp.class);

}

*Purpose:* Connects the preferences storage to the application package.  
*Modification:* Changing the package here isn’t common, but if you want to use a different node, modify this line.

* **Getters & Setters:**  
  Methods such as getGlobalVolume(), isDarkMode(), getInitialTime(), etc., retrieve stored values.  
  For example, to change the default initial time (used for quiz timers), change:

java

CopyEdit

public static int getInitialTime() {

return prefs.getInt("initialTime", 60);

}

*Modification:* Replace 60 with another default value (e.g., 120 for 2 minutes).

* **Difficulty Settings:**  
  The method setDifficulty(int seconds, String levelName) sets both the time and prints the level name.  
  *Modification:* You can adjust the number of seconds and text labels for each difficulty level.
* **Sound Setting:**  
  The methods getSoundEnabled() and setSoundEnabled(boolean enabled) control whether sound effects play.  
  *Modification:* Changing the default in getSoundEnabled() (default true) is possible if needed.

**2. AdventureMode.java**

**Purpose:**  
Creates an interactive 2D adventure game scene with basic movement and collision detection. When the player reaches a special area, a quiz popup is triggered.

**Code Explanation & Modification Tips:**

* **UI Layout & Styling:**  
  The game uses a Pane with a light green background and adds a blue rectangle for the player, a gray rectangle for a wall, and a hidden button (liftBox) that becomes visible on collision.  
  *Modification:*
  + Change background color by editing:

java

CopyEdit

gamePane.setStyle("-fx-background-color: lightgreen;");

* + Modify the player’s color or size by changing:

java

CopyEdit

Rectangle player = new Rectangle(40, 40, Color.BLUE);

* **Movement & Collision:**  
  Key events (LEFT/RIGHT) control a horizontal movement variable (dx). Collision with the wall is detected, and the liftBox button appears.  
  *Modification:*
  + To change movement speed, adjust the value assigned to dx[0] (e.g., -5 or 5).
  + Change collision boundaries by modifying the condition that compares player position to the wall’s layout.
* **Quiz Pop-Up:**  
  When the liftBox button is pressed, the timer stops and showAdventureQuestion() is called to display a quiz question in a new Stage.  
  *Modification:*
  + To change the question text or options, modify the labels and button texts inside the showAdventureQuestion() method.
  + You can adjust the size and styling of the quiz popup by changing the VBox’s padding, alignment, or scene dimensions.

**3. AudioManager.java**

**Purpose:**  
Handles audio playback for sound effects (applause, wrong answer, tile flip) using JavaFX’s AudioClip.

**Code Explanation & Modification Tips:**

* **Initialization:**  
  Loads sound files from resources using their paths.

java

CopyEdit

URL applauseURL = AudioManager.class.getResource("/sounds/applause.mp3");

*Modification:*

* + To change the sound, update the file path or replace the audio file in your resources.
  + Adjust default volume by changing the parameter in applauseClip.setVolume(PreferencesManager.getGlobalVolume());.
* **Playback Methods:**  
  Methods like playApplause(), playWrong(), and playTileFlip() check if sound is enabled and play the corresponding clip.  
  *Modification:*
  + To change playback behavior (e.g., loop a sound), modify the AudioClip settings.
* **Volume Updates:**  
  updateVolume(double volume) sets the volume for each audio clip. *Modification:*
  + Changing volume effects requires updating this method if you want different volumes for different sounds.

**4. ClassicModeSelectionPopup.java**

**Purpose:**  
Provides a pop-up for selecting between different classic game modes (Classic Quiz, Matching Titles Game, Retro Runner or Fighting).

**Code Explanation & Modification Tips:**

* **Modal Layer:**  
  Uses a StackPane overlay with a semi-transparent blocker rectangle.  
  *Modification:*
  + To change the overlay color or opacity, modify:

java

CopyEdit

blocker.setFill(Color.rgb(0, 0, 0, 0.3));

* **UI Components:**  
  Contains a BorderPane with a title and two buttons.  
  *Modification:*
  + Change the title text or style by editing the Label title.
  + Modify button styles (font size, padding, background color) by editing the style string in buttonStyle.
  + To adjust button animations, modify the ScaleTransition durations (e.g., change Duration.millis(150)).
* **Navigation:**  
  Each button’s action calls a different scene (via SceneManager.showQuizScene() or SceneManager.showMatchingTitlesScene()). *Modification:*
  + To change navigation logic, update the event handler for each button.

**5. FullQuestionPopup.java**

**Purpose:**  
Shows a popup window with the full text of a question when needed.

**Code Explanation & Modification Tips:**

* **Layout:**  
  Uses a VBox for a simple vertical layout with a wrapping label for the question and a “Close” button.  
  *Modification:*
  + Change window size by editing:

java

CopyEdit

Scene scene = new Scene(layout, 400, 300);

* + To modify font size or text color, adjust the style of questionLabel (e.g., -fx-font-size: 16px;).
* **Popup Behavior:**  
  The Stage is set to modal so that the user must close it before returning to the main window. *Modification:*
  + To allow non-modal behavior, remove or change popupStage.initModality(Modality.APPLICATION\_MODAL);.

**6. LeaderboardEntry.java**

**Purpose:**  
Defines a data model for an individual leaderboard entry using JavaFX properties, making it compatible with TableView controls.

**Code Explanation & Modification Tips:**

* **Properties:**  
  Contains SimpleStringProperty for name, time, and mode, and a SimpleIntegerProperty for score.  
  *Modification:*
  + To add additional fields (like a date or rank), add new properties following the same pattern.
  + To change the default formatting in the UI, you might adjust the TableView cell factories in the scene that displays these entries.

**7. LeaderboardScene.java**

**Purpose:**  
Displays the leaderboard screen with animated effects, a custom background, and sorted leaderboard entries.

**Code Explanation & Modification Tips:**

* **Background Layer:**  
  Combines a particle effect pane, a background image, and a dark overlay.  
  *Modification:*
  + Change the background image by replacing the resource path in:

java

CopyEdit

LeaderboardScene.class.getResource("/images/arena\_bg.jpeg").toExternalForm()

* + Adjust overlay opacity by editing the Color.color(0, 0, 0, 0.4) value.
* **Title and Animations:**  
  A large title is displayed with drop shadows and a fade-in effect.  
  *Modification:*
  + Change font family, size, or color by modifying the setFont(Font.font("Verdana", 48)) and style strings.
  + To alter fade transition duration, change Duration.seconds(1.2).
* **Leaderboard Data & Sorting:**  
  Loads data via LeaderboardService.loadLeaderboard(), sorts it (including custom ranking for difficulty), and displays the top three with special styling.  
  *Modification:*
  + To change sorting criteria, adjust the comparator in the stream.
  + To change border colors or spacing in the top three cards, update the CSS style strings.
* **Remaining Entries:**  
  A scrollable VBox lists all remaining entries with grid layouts.  
  *Modification:*
  + Adjust column constraints, gaps, and font sizes to modify the layout.
* **Navigation:**  
  A right arrow button navigates to a (placeholder) adventure leaderboard, and a bottom “Back to Main Menu” button returns to the main menu. *Modification:*
  + Update button texts, sizes, and hover effects by modifying the style and animation transitions.

**8. LeaderboardService.java**

**Purpose:**  
Handles communication with a remote database (Supabase) for pushing new leaderboard data and loading the leaderboard entries.

**Code Explanation & Modification Tips:**

* **API Communication:**  
  Uses HttpClient to send GET requests to fetch leaderboard data and POST/PATCH requests to update scores.  
  *Modification:*
  + Change the API URL or API key by modifying the constants SUPABASE\_API\_URL and SUPABASE\_API\_KEY.
  + To adjust the JSON field names (if your database schema changes), update the code that extracts fields from the JSON response.
* **Data Parsing:**  
  Uses Gson and JsonParser to convert JSON data into Java objects.  
  *Modification:*
  + To handle new fields in the leaderboard, update the parsing logic and modify the constructor call for LeaderboardEntry.

**9. MainMenu.java**

**Purpose:**  
Creates the main menu UI for the app with navigation buttons to different game modes and settings.

**Code Explanation & Modification Tips:**

* **UI Layout:**  
  Uses a VBox to list buttons such as “Classic Mode”, “Adventure Mode”, “Leaderboard”, “Settings”, and “Quit”.  
  *Modification:*
  + Change button texts or add icons by modifying the strings (e.g., "🎮 Classic Mode").
  + To change the spacing between buttons, adjust the spacing value in new VBox(30, ...).
  + Modify overall font sizes and styles in the title label via:

java

CopyEdit

title.setStyle("-fx-font-size: 48px; -fx-font-weight: bold;");

* **Button Animations:**  
  Each button uses a ScaleTransition to slightly enlarge on mouse hover.  
  *Modification:*
  + To change the animation speed, update the duration in Duration.millis(150).
* **Navigation Handlers:**  
  Button click events call methods in SceneManager to switch scenes. *Modification:*
  + To change which scene a button navigates to, update the lambda in the setOnAction() method.

**10. MatchingTitlesScene.java**

**Purpose:**  
Implements a matching game where players pair question cards with their corresponding answer cards (plus distractor cards).

**Code Explanation & Modification Tips:**

* **Game Setup:**  
  Loads questions via QuestionService.loadQuestionsWithCache(), creates a list of cards (of types QUESTION, ANSWER, and DISTRACTOR), and shuffles them. *Modification:*
  + To change the number of distractor cards, adjust the logic that creates distractorCards.
  + To modify card text, change the code that extracts a title from the question via the extractTitle() method.
* **Layout:**  
  Uses a GridPane for the card layout. The number of columns is set (e.g., 6 columns). *Modification:*
  + Adjust horizontal/vertical gaps or padding in the grid by editing grid.setHgap(10) or grid.setPadding(new Insets(20)).
* **Timer & Score:**  
  A timer is started using a Timeline with the initial time pulled from PreferencesManager.getInitialTime(). The score is updated based on matched pairs. *Modification:*
  + To change the timer duration, modify the initial value stored in preferences or change the logic in startTimer().
  + To modify score calculation, change the multiplication factor in:

java

CopyEdit

score = matchedCount \* 10;

* **Card Interactions:**  
  Inner classes (like Card and QnAInfo) manage individual card behavior, including flip animations and match checking. *Modification:*
  + Change card size and style by editing view.setPrefSize(300, 180) and the CSS in view.setStyle().
  + To adjust animation speed, modify the duration in the FadeTransition.
* **Revision Pane:**  
  At game end, a revision pane is shown listing unmatched questions along with their hints, keywords, and explanations. *Modification:*
  + Change the appearance of the revision pane by updating the VBox’s style and label fonts.

**11. SceneManager.java**

**Purpose:**  
Centralizes scene switching within the app. It holds a single Scene and allows all screens to update the root node for navigation.

**Code Explanation & Modification Tips:**

* **Initialization:**

java

CopyEdit

public static void initialize(Stage stage) {

primaryStage = stage;

mainScene = new Scene(new StackPane(), 800, 600);

primaryStage.setScene(mainScene);

primaryStage.setFullScreen(true);

primaryStage.setFullScreenExitHint("");

primaryStage.show();

}

*Modification:*

* + To change window size, update the dimensions in the new Scene(new StackPane(), 800, 600).
  + To disable full-screen mode, remove or modify primaryStage.setFullScreen(true).
* **Theme Application:**  
  After updating the scene root with setRoot(Parent root), the theme is re-applied using ThemeManager.applyTheme(mainScene). *Modification:*
  + Adjusting themes is done in the ThemeManager; changes here affect all scenes.
* **Navigation Methods:**  
  Methods like showMainMenu(), showQuizScene(), etc., update the scene’s root to the respective UI component. *Modification:*
  + To change navigation, modify the call inside each method (e.g., which scene is loaded when a button is pressed).

**12. SettingsScene.java**

**Purpose:**  
Provides a modal overlay for users to adjust gameplay (difficulty), themes, and sound settings.

**Code Explanation & Modification Tips:**

* **Overlay Layout:**  
  Uses a BorderPane with a fixed size and dark semi-transparent background. *Modification:*
  + To change overlay size, modify overlayPane.setPrefSize(400, 400) and its max size.
  + Change background color or opacity in the style string (e.g., modify rgba(0,0,0,0.8)).
* **Gameplay Settings:**  
  Contains a hidden HBox that shows “Classic” and “Adventure” buttons on hover, with a context menu for difficulty selection.  
  *Modification:*
  + To adjust difficulty durations, change the numbers in the actions (e.g., 600 for Amateur).
  + Change button styles or hover behavior by updating the style and event handlers.
* **Theme Settings:**  
  Displays buttons for “Light”, “Dark”, and “Custom”.  
  *Modification:*
  + Change the theme names or the actions that update preferences.
  + To modify custom file selection, update the FileChooser extension filter if you want more file types.
* **Sound Toggle:**  
  A ToggleButton shows “ON” or “OFF” based on sound settings. *Modification:*
  + To change the appearance, update the styles in updateSoundToggleStyle().
* **Close Button:**  
  Returns to the main menu when clicked. *Modification:*
  + Change text or styling of the “Close” button by editing its style and event handler.

**13. ThemeManager.java**

**Purpose:**  
Handles applying CSS-based themes to the current scene based on user preferences.

**Code Explanation & Modification Tips:**

* **applyTheme(Scene scene):**  
  Checks if dark mode is enabled and if a custom background is set. It then builds a CSS style string and applies it to the scene’s root. *Modification:*
  + To change the default light/dark colors, edit the CSS strings:

java

CopyEdit

themeStyle = "-fx-background-color: #333333; -fx-text-fill: white;";

or for light mode change to a different background color.

* + To adjust the custom background behavior, modify the concatenated string that sets -fx-background-image.
* **applyThemeToAllScenes():**  
  Simply reapplies the theme to the scene managed by SceneManager.

**14. VTUGamifiedQuizApp.java**

**Purpose:**  
The main application class that launches the JavaFX app. It initializes preferences, audio, and scene management, and displays the main menu.

**Code Explanation & Modification Tips:**

* **start(Stage primaryStage):**  
  Calls initialization methods for PreferencesManager and AudioManager. Then initializes SceneManager and shows the main menu. *Modification:*
  + To change startup behavior (for example, to start in a different scene), modify the call to SceneManager.showMainMenu().
* **main(String[] args):**  
  Standard JavaFX application launch. *Modification:*
  + Command-line arguments can be used if additional startup logic is needed.
* **Compilation/Execution Comments:**  
  Instructions for compiling and running the app (including module paths for JavaFX, Gson, and LWJGL) are included as comments. *Modification:*
  + Update these instructions if you change library paths or add new modules.

**15. WrongAnswerRecord.java**

**Purpose:**  
A simple data model for recording details of questions answered incorrectly. It stores the question, the selected answer, the correct answer, a hint, keywords, and an explanation.

**Code Explanation & Modification Tips:**

* **Constructor & Getters:**  
  All fields are set through the constructor and accessed via getter methods. *Modification:*
  + To add additional fields (e.g., date or category), update the constructor and add corresponding getters.
  + To change how these records are displayed, modify the UI in QuizScene where they are used.

**16. Question.java**

**Purpose:**  
Represents a quiz question. This class stores the question text, answer options, the index of the correct answer, a hint, keywords, and an explanation.

**Code Explanation & Modification Tips:**

* **Fields & Constructor:**  
  All properties are provided during object construction.  
  *Modification:*
  + To change default behavior (for instance, if you want to add an extra option), you’d update the constructor and any methods that rely on the options array.
  + To modify the way hints or explanations are stored, adjust the corresponding fields.
* **Getters:**  
  Provide access to all stored information. *Modification:*
  + Additional logic (e.g., formatting) can be added if needed when returning values.

**17. QuestionService.java**

**Purpose:**  
Loads questions from a remote API (Supabase) and caches them locally. Provides fallback hardcoded questions if remote loading fails.

**Code Explanation & Modification Tips:**

* **Remote Loading:**  
  Uses Java’s HttpClient to fetch JSON from a specified Supabase API URL.  
  *Modification:*
  + Update the constant QUESTIONBANK\_API\_URL or the API key if your database settings change.
  + If the JSON schema changes (e.g., field names like "optionA"), update the parsing logic accordingly.
* **Caching:**  
  Reads from and writes to a local file (questions\_cache.json) and uses MD5 hashing to determine if the remote questions differ from the cache. *Modification:*
  + To change the cache filename, update the file name in loadCachedQuestions() and saveQuestionsToCache().
* **Parsing & Fallback:**  
  Converts JSON entries to Question objects. If no questions load, it calls initializeHardcodedQuestions().  
  *Modification:*
  + To add more fallback questions or adjust existing ones, edit the list in initializeHardcodedQuestions().

**18. QuizScene.java**

**Purpose:**  
Implements the classic quiz mode where a question is presented with multiple-choice options. It manages the timer, score, wrong answer recording, and even shows a hint popup.

**Code Explanation & Modification Tips:**

* **Layout & UI Components:**  
  The quiz scene is built with a VBox containing labels (for timer, score, question) and buttons for each option.  
  *Modification:*
  + Change spacing, padding, and background color by modifying the style in quizLayout.setStyle().
  + To change the font or text size of labels and buttons, update their style strings (e.g., -fx-font-size: 18px;).
* **Timer & Score Management:**  
  Uses a Timeline (quizTimer) to decrement timeRemaining every second.  
  *Modification:*
  + To change the countdown speed or start time, update timeRemaining (usually set via PreferencesManager) or modify the Timeline’s KeyFrame duration.
  + Score increases by 10 for each correct answer. Modify the line score += 10; if a different scoring system is desired.
* **Answer Handling:**  
  When an option button is clicked, it checks whether the selected option is correct, updates the score, plays audio feedback, and records wrong answers (using WrongAnswerRecord).  
  *Modification:*
  + To change the feedback text or sound effects, update the corresponding code within the event handler.
  + To change the order of questions, review the method getRandomQuestion() which removes the first question from the list.
* **Hint Popup:**  
  A hint icon (light bulb image) is added and animated with a pulsating effect. Hovering over it shows a small Popup with the hint text.  
  *Modification:*
  + Change the icon by updating the resource path in:

java

CopyEdit

new Image(QuizScene.class.getResourceAsStream("/images/bulb.png"));

* + To modify the popup’s appearance (colors, padding), update the style in the VBox that is added to the Popup.
  + Adjust animation speed by modifying the duration in ScaleTransition.
* **End of Quiz:**  
  When time expires, endQuiz() is called, which creates a summary screen with options to save the score, play again, return to the main menu, or quit.  
  *Modification:*
  + To change the layout of the end screen, adjust the BorderPane or VBox properties.
  + Change the styling of the revision pane (which lists wrong answers) by modifying the CSS in createWrongAnswersPane().

**Final Remarks**

This document provides a complete walkthrough of all 18 Java files. It covers:

* **Code Explanation:** What each part of the code does and how the files interact.
* **Purpose:** Why the file exists and what functionality it provides.
* **Modification Guidance:** Clear instructions on how to change attributes like time, speed, colors, fonts, text sizes, and overall UI layouts in each window.

With this documentation, even a beginner should be able to understand the structure of the Maze of Knowledge App and how to tweak various elements to suit different design requirements or gameplay mechanics.

Feel free to refer back to each section when making changes, and let this guide serve as a roadmap for further customizations or enhancements to your application.