```
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';

import 'app_localizations_ar.dart';
import 'app_localizations_bn.dart';
import 'app_localizations_de.dart';
import 'app_localizations_en.dart';
import 'app_localizations_es.dart';
import 'app_localizations_fr.dart';
import 'app_localizations_hi.dart';
import 'app_localizations_id.dart';
import 'app_localizations_it.dart';
import 'app_localizations_ja.dart';
import 'app_localizations_ko.dart';
import 'app_localizations_pt.dart';
import 'app_localizations_ru.dart';
import 'app_localizations_th.dart';
import 'app_localizations_tr.dart';
import 'app_localizations_ur.dart';
import 'app_localizations_vi.dart';
import 'app_localizations_zh.dart';

/// 全球前18名语言（按使用人数）
const List&lt;String&gt; _supportedLanguageCodes = [
  'en', 'zh', 'hi', 'es', 'ar', 'fr', 'bn', 'pt', 'ru', 'id',
  'ur', 'de', 'ja', 'vi', 'tr', 'ko', 'it', 'th',
];

abstract class AppLocalizations {
  static AppLocalizations of(BuildContext context) {
    return Localizations.of&lt;AppLocalizations&gt;(context, AppLocalizations)!;
  }

  static const LocalizationsDelegate&lt;AppLocalizations&gt; delegate =
      _AppLocalizationsDelegate();

  static final Map&lt;String, AppLocalizations&gt; _localizations = {
    'en': AppLocalizationsEn(),
    'zh': AppLocalizationsZh(),
    'hi': AppLocalizationsHi(),
    'es': AppLocalizationsEs(),
    'ar': AppLocalizationsAr(),
    'fr': AppLocalizationsFr(),
    'bn': AppLocalizationsBn(),
    'pt': AppLocalizationsPt(),
    'ru': AppLocalizationsRu(),
    'id': AppLocalizationsId(),
    'ur': AppLocalizationsUr(),
    'de': AppLocalizationsDe(),
    'ja': AppLocalizationsJa(),
```

```
  'vi': AppLocalizationsVi(),
  'tr': AppLocalizationsTr(),
  'ko': AppLocalizationsKo(),
  'it': AppLocalizationsIt(),
  'th': AppLocalizationsTh(),
};

static AppLocalizations get(String locale) {
  final code = locale.split('-').first.toLowerCase();
  if (code == 'zh') return _localizations['zh']!;
  return _localizations[code] ?? _localizations['en']!;
}

// 通用
String get appTitle;
String get sudoku;
String get loading;

// 首页
String get slogan;
String get freeMode;
String get campaignMode;
String get rulesTitle;

// 规则说明
String get rulesGoal;
String get rulesGoalDesc;
String get rulesBasic;
String get rulesBasicItems;
String get rulesOps;
String get rulesOpsItems;
String get rulesGotIt;

// 游戏（带参数）
String level(int n) =&gt; levelFormat.replaceAll('%s', n.toString());
String errors(int count, int max) =&gt;
    errorsFormat.replaceFirst('%s', count.toString()).replaceFirst('%s', max.toString());
String gameOverDescFormatted(int max, String time) =&gt;
    gameOverDesc.replaceFirst('%s', max.toString()).replaceFirst('%s', time);
String congratsDescFormatted(String time) =&gt;
    congratsDesc.replaceFirst('%s', time);
String startLevelFormatted(int n) =&gt; startLevel.replaceAll('%s', n.toString());
String unlockedToFormatted(int n) =&gt; unlockedTo.replaceFirst('%s', n.toString());
String unlockedToClickToStartFormatted(int n) =&gt;
    unlockedToClickToStart.replaceFirst('%s', n.toString());

String get levelFormat;
String get errorsFormat;
String get gameOver;
String get gameOverDesc;
```

```
  String get startLevel;
  String get unlockedTo;
  String get back;
  String get retry;
  String get congrats;
  String get congratsDesc;
  String get confirm;
  String get nextLevel;
  String get newGame;
  String get selectDifficulty;
  String get difficultyEasy;
  String get difficultyMedium;
  String get difficultyHard;
  String get replayLevel;
  String get difficulty;

  // 闯关
  String get campaignTitle;
  String get campaignSubtitle;
  String get modeEasy;
  String get modeMedium;
  String get modeHard;
  String get modeLevels;
  String get clickToStart;
  String get unlockedToClickToStart;
}

class _AppLocalizationsDelegate
    extends LocalizationsDelegate&lt;AppLocalizations&gt; {
  const _AppLocalizationsDelegate();

  @override
  bool isSupported(Locale locale) =&gt;
      _supportedLanguageCodes.contains(locale.languageCode.toLowerCase());

  @override
  Future&lt;AppLocalizations&gt; load(Locale locale) async =&gt;
      SynchronousFuture&lt;AppLocalizations&gt;(AppLocalizations.get(locale.languageCode));

  @override
  bool shouldReload(_AppLocalizationsDelegate old) =&gt; false;
}
import 'app_localizations.dart';

class AppLocalizationsAr extends AppLocalizations {
  @override
  String get appTitle =&gt; 'CS وكودوس';

  @override
  String get sudoku =&gt; 'وكودوس';
```

```
@override
String get loading =&gt; 'جاري التحميل...';

@override
String get slogan =&gt; 'لعبة المنطق الرقمية الكلاسيكية';

@override
String get freeMode =&gt; 'اللعب الحر';

@override
String get campaignMode =&gt; 'الحملة';

@override
String get rulesTitle =&gt; 'وقواعد السودوكو';

@override
String get rulesGoal =&gt; 'الهدف';

@override
String get rulesGoalDesc =&gt;
    'املأ شبكة 9×9 بالأرقام من 1 إلى 9 بحيث يحتوي كل صف وعمود ومربع 3×3 '
    'على كل رقم مرة واحدة بالضبط.';

@override
String get rulesBasic =&gt; 'القواعد';

@override
String get rulesBasicItems =&gt;
    '• كل صف لك يف رارتك ال\n'
    '• كل عمود لك يف رارتك ال\n'
    '• كل مربع 3×3 لك يف رارتك ال\n'
    '• الأرقام المطاعمة ال يمكن تغييرها';

@override
String get rulesOps =&gt; 'كيفية اللعب';

@override
String get rulesOpsItems =&gt;
    '• انقر على خلية للتحديد، ثم انقر على رقم للتعبئة\n'
    '• دودر فع لع وفرية: أخضر = صحيح، أحمر = خطأ\n'
    '• وضع الحملة له حد 10 أخطاء';

@override
String get rulesGotIt =&gt; 'فهمت';

@override
String get levelFormat =&gt; 'المستوى %s';

@override
```

```
String get errorsFormat =&gt; 'الأخطاء: %s/%s';

@override
String get gameOver =&gt; 'انتهت اللعبة';

@override
String get gameOverDesc =&gt; 'وصلت إلى %s أخطاء. الوقت: %s';

@override
String get back =&gt; 'رجوع';

@override
String get retry =&gt; 'إعادة المحاولة';

@override
String get congrats =&gt; 'تهانينا!';

@override
String get congratsDesc =&gt; 'أكملت في %s!';

@override
String get confirm =&gt; 'موافق';

@override
String get nextLevel =&gt; 'التالي';

@override
String get newGame =&gt; 'لعبة جديدة';

@override
String get selectDifficulty =&gt; 'اختر الصعوبة';

@override
String get difficultyEasy =&gt; 'سهل';

@override
String get difficultyMedium =&gt; 'متوسط';

@override
String get difficultyHard =&gt; 'صعب';

@override
String get replayLevel =&gt; 'إعادة اللعب';

@override
String get difficulty =&gt; 'الصعوبة';

@override
String get campaignTitle =&gt; 'الحملة';
```

```
  @override
  String get campaignSubtitle =&gt; '999 مستوى، تزداد الصعوبة كل 9 مستويات';

  @override
  String get modeEasy =&gt; 'سهل';

  @override
  String get modeMedium =&gt; 'متوسط';

  @override
  String get modeHard =&gt; 'صعب';

  @override
  String get modeLevels =&gt; '999 مستوى';

  @override
  String get startLevel =&gt; 'بدء المستوى %s';

  @override
  String get unlockedTo =&gt; 'مفتوح حتى المستوى %s';

  @override
  String get clickToStart =&gt; 'انقر للبدء';

  @override
  String get unlockedToClickToStart =&gt; 'مفتوح حتى المستوى %s. انقر للبدء';
}
import 'app_localizations.dart';

class AppLocalizationsBn extends AppLocalizations {
  @override
  String get appTitle =&gt; 'CS সুডোকু';

  @override
  String get sudoku =&gt; 'সুডোকু';

  @override
  String get loading =&gt; 'লোড হচ্ছে...';

  @override
  String get slogan =&gt; 'ক্লাসিক সংখ্যা লজিক গেম';

  @override
  String get freeMode =&gt; 'ফ্রি প্লে';

  @override
  String get campaignMode =&gt; 'ক্যাম্পেইন';

  @override
  String get rulesTitle =&gt; 'সুডোকু নিয়ম';
```

```
@override
String get rulesGoal =&gt; 'লক্ষ্য';

@override
String get rulesGoalDesc =&gt;
    '৯×৯ গ্রিডে ১-৯ সংখ্যা দিয়ে পূরণ করুন যাতে প্রতিটি সারি, কলাম এবং '
    '৩×৩ বক্সে প্রতিটি সংখ্যা ঠিক একবার থাকে।';

@override
String get rulesBasic =&gt; 'নিয়ম';

@override
String get rulesBasicItems =&gt;
    '• প্রতিটি সারিতে পুনরাবৃত্তি নেই\n'
    '• প্রতিটি কলামে পুনরাবৃত্তি নেই\n'
    '• প্রতিটি ৩×৩ বক্সে পুনরাবৃত্তি নেই\n'
    '• দেওয়া সংখ্যা পরিবর্তন করা যাবে না';

@override
String get rulesOps =&gt; 'কিভাবে খেলবেন';

@override
String get rulesOpsItems =&gt;
    '• নির্বাচন করতে একটি সেল ট্যাপ করুন, তারপর পূরণ করতে একটি সংখ্যা ট্যাপ করুন\n'
    '• তাৎক্ষণিক প্রতিক্রিয়া: সবুজ = সঠিক, লাল = ভুল\n'
    '• ক্যাম্পেইন মোডে ১০টি ভুলের সীমা';

@override
String get rulesGotIt =&gt; 'বুঝেছি';

@override
String get levelFormat =&gt; 'লেভেল %s';

@override
String get errorsFormat =&gt; 'ভুল: %s/%s';

@override
String get gameOver =&gt; 'গেম ওভার';

@override
String get gameOverDesc =&gt; 'আপনি %sটি ভুল করেছেন। সময়: %s';

@override
String get back =&gt; 'পিছনে';

@override
String get retry =&gt; 'আবার চেষ্টা করুন';

@override
```

```
String get congrats =&gt; 'অভিনন্দন!';

@override
String get congratsDesc =&gt; '%s এ সম্পন্ন!';

@override
String get confirm =&gt; 'ঠিক আছে';

@override
String get nextLevel =&gt; 'পরবর্তী';

@override
String get newGame =&gt; 'নতুন গেম';

@override
String get selectDifficulty =&gt; 'কঠিনতা নির্বাচন করুন';

@override
String get difficultyEasy =&gt; 'সহজ';

@override
String get difficultyMedium =&gt; 'মাঝারি';

@override
String get difficultyHard =&gt; 'কঠিন';

@override
String get replayLevel =&gt; 'পুনরায় খেলুন';

@override
String get difficulty =&gt; 'কঠিনতা';

@override
String get campaignTitle =&gt; 'ক্যাম্পেইন';

@override
String get campaignSubtitle =&gt; '৯৯৯ লেভেলে, প্রতি ৯ লেভেলে কঠিনতা বাড়ে';

@override
String get modeEasy =&gt; 'সহজ';

@override
String get modeMedium =&gt; 'মাঝারি';

@override
String get modeHard =&gt; 'কঠিন';

@override
String get modeLevels =&gt; '৯৯৯ লেভেল';
```

```
  @override
  String get startLevel =&gt; 'লভেলে %s শুরু করুন';

  @override
  String get unlockedTo =&gt; 'লভেলে %s পর্যন্ত আনলক';

  @override
  String get clickToStart =&gt; 'শুরু করতে ট্যাপ করুন';

  @override
  String get unlockedToClickToStart =&gt; 'লভেলে %s পর্যন্ত আনলক। শুরু করতে ট্যাপ করুন';
}
import 'app_localizations.dart';

class AppLocalizationsDe extends AppLocalizations {
  @override
  String get appTitle =&gt; 'CS Sudoku';

  @override
  String get sudoku =&gt; 'Sudoku';

  @override
  String get loading =&gt; 'Laden...';

  @override
  String get slogan =&gt; 'Klassisches Zahlenlogik-Spiel';

  @override
  String get freeMode =&gt; 'Freies Spiel';

  @override
  String get campaignMode =&gt; 'Kampagne';

  @override
  String get rulesTitle =&gt; 'Sudoku-Regeln';

  @override
  String get rulesGoal =&gt; 'Ziel';

  @override
  String get rulesGoalDesc =&gt;
      'Füllen Sie das 9×9-Raster mit den Ziffern 1–9, sodass jede Zeile, '
      'Spalte und jedes 3×3-Feld jede Zahl genau einmal enthält.';

  @override
  String get rulesBasic =&gt; 'Regeln';

  @override
  String get rulesBasicItems =&gt;
      '• Keine Wiederholungen in jeder Zeile\n'
```

```
    '• Keine Wiederholungen in jeder Spalte\n'
    '• Keine Wiederholungen in jedem 3×3-Feld\n'
    '• Vorgegebene Zahlen können nicht geändert werden';

@override
String get rulesOps =&gt; 'Spielanleitung';

@override
String get rulesOpsItems =&gt;
    '• Tippen Sie auf eine Zelle zur Auswahl, dann auf eine Zahl zum Ausfüllen\n'
    '• Sofortiges Feedback: Grün = richtig, Rot = falsch\n'
    '• Kampagnenmodus hat ein Limit von 10 Fehlern';

@override
String get rulesGotIt =&gt; 'Verstanden';

@override
String get levelFormat =&gt; 'Level %s';

@override
String get errorsFormat =&gt; 'Fehler: %s/%s';

@override
String get gameOver =&gt; 'Spiel vorbei';

@override
String get gameOverDesc =&gt; 'Sie haben %s Fehler erreicht. Zeit: %s';

@override
String get back =&gt; 'Zurück';

@override
String get retry =&gt; 'Erneut versuchen';

@override
String get congrats =&gt; 'Herzlichen Glückwunsch!';

@override
String get congratsDesc =&gt; 'Abgeschlossen in %s!';

@override
String get confirm =&gt; 'OK';

@override
String get nextLevel =&gt; 'Weiter';

@override
String get newGame =&gt; 'Neues Spiel';

@override
```

```
  String get selectDifficulty =&gt; 'Schwierigkeit wählen';

  @override
  String get difficultyEasy =&gt; 'Einfach';

  @override
  String get difficultyMedium =&gt; 'Mittel';

  @override
  String get difficultyHard =&gt; 'Schwer';

  @override
  String get replayLevel =&gt; 'Wiederholen';

  @override
  String get difficulty =&gt; 'Schwierigkeit';

  @override
  String get campaignTitle =&gt; 'Kampagne';

  @override
  String get campaignSubtitle =&gt; '999 Level, Schwierigkeit steigt alle 9 Level';

  @override
  String get modeEasy =&gt; 'Einfach';

  @override
  String get modeMedium =&gt; 'Mittel';

  @override
  String get modeHard =&gt; 'Schwer';

  @override
  String get modeLevels =&gt; '999 Level';

  @override
  String get startLevel =&gt; 'Level %s starten';

  @override
  String get unlockedTo =&gt; 'Freigeschaltet bis Level %s';

  @override
  String get clickToStart =&gt; 'Tippen zum Starten';

  @override
  String get unlockedToClickToStart =&gt; 'Freigeschaltet bis Level %s. Tippen zum Starten';
}
import 'app_localizations.dart';

class AppLocalizationsEn extends AppLocalizations {
```

```
@override
String get appTitle =&gt; 'CS Sudoku';

@override
String get sudoku =&gt; 'Sudoku';

@override
String get loading =&gt; 'Loading...';

@override
String get slogan =&gt; 'Classic Number Logic Game';

@override
String get freeMode =&gt; 'Free Play';

@override
String get campaignMode =&gt; 'Campaign';

@override
String get rulesTitle =&gt; 'Sudoku Rules';

@override
String get rulesGoal =&gt; 'Goal';

@override
String get rulesGoalDesc =&gt;
    'Fill the 9×9 grid with digits 1–9 so that each row, column, '
    'and 3×3 box contains each number exactly once.';

@override
String get rulesBasic =&gt; 'Rules';

@override
String get rulesBasicItems =&gt;
    '• No repeats in each row\n'
    '• No repeats in each column\n'
    '• No repeats in each 3×3 box\n'
    '• Given numbers cannot be changed';

@override
String get rulesOps =&gt; 'How to Play';

@override
String get rulesOpsItems =&gt;
    '• Tap a cell to select, then tap a number to fill\n'
    '• Instant feedback: green = correct, red = wrong\n'
    '• Campaign mode has a 10-error limit';

@override
String get rulesGotIt =&gt; 'Got it';
```

```
@override
String get levelFormat =&gt; 'Level %s';

@override
String get errorsFormat =&gt; 'Errors: %s/%s';

@override
String get gameOver =&gt; 'Game Over';

@override
String get gameOverDesc =&gt; 'You reached %s errors. Time: %s';

@override
String get back =&gt; 'Back';

@override
String get retry =&gt; 'Retry';

@override
String get congrats =&gt; 'Congratulations!';

@override
String get congratsDesc =&gt; 'Completed in %s!';

@override
String get confirm =&gt; 'OK';

@override
String get nextLevel =&gt; 'Next';

@override
String get newGame =&gt; 'New Game';

@override
String get selectDifficulty =&gt; 'Select Difficulty';

@override
String get difficultyEasy =&gt; 'Easy';

@override
String get difficultyMedium =&gt; 'Medium';

@override
String get difficultyHard =&gt; 'Hard';

@override
String get replayLevel =&gt; 'Replay';

@override
```

```
  String get difficulty =&gt; 'Difficulty';

  @override
  String get campaignTitle =&gt; 'Campaign';

  @override
  String get campaignSubtitle =&gt; '999 levels, difficulty increases every 9 levels';

  @override
  String get modeEasy =&gt; 'Easy';

  @override
  String get modeMedium =&gt; 'Medium';

  @override
  String get modeHard =&gt; 'Hard';

  @override
  String get modeLevels =&gt; '999 levels';

  @override
  String get startLevel =&gt; 'Start Level %s';

  @override
  String get unlockedTo =&gt; 'Unlocked to Level %s';

  @override
  String get clickToStart =&gt; 'Tap to start';

  @override
  String get unlockedToClickToStart =&gt; 'Unlocked to Level %s. Tap to start';
}
import 'app_localizations.dart';

class AppLocalizationsEs extends AppLocalizations {
  @override
  String get appTitle =&gt; 'CS Sudoku';

  @override
  String get sudoku =&gt; 'Sudoku';

  @override
  String get loading =&gt; 'Cargando...';

  @override
  String get slogan =&gt; 'Juego clásico de lógica numérica';

  @override
  String get freeMode =&gt; 'Juego libre';
```

```
@override
String get campaignMode =&gt; 'Campaña';

@override
String get rulesTitle =&gt; 'Reglas del Sudoku';

@override
String get rulesGoal =&gt; 'Objetivo';

@override
String get rulesGoalDesc =&gt;
    'Completa la cuadrícula 9×9 con dígitos del 1 al 9 de modo que cada fila, '
    'columna y caja 3×3 contenga cada número exactamente una vez.';

@override
String get rulesBasic =&gt; 'Reglas';

@override
String get rulesBasicItems =&gt;
    '• Sin repeticiones en cada fila\n'
    '• Sin repeticiones en cada columna\n'
    '• Sin repeticiones en cada caja 3×3\n'
    '• Los números dados no se pueden cambiar';

@override
String get rulesOps =&gt; 'Cómo jugar';

@override
String get rulesOpsItems =&gt;
    '• Toca una celda para seleccionar, luego toca un número para rellenar\n'
    '• Retroalimentación instantánea: verde = correcto, rojo = incorrecto\n'
    '• El modo campaña tiene un límite de 10 errores';

@override
String get rulesGotIt =&gt; 'Entendido';

@override
String get levelFormat =&gt; 'Nivel %s';

@override
String get errorsFormat =&gt; 'Errores: %s/%s';

@override
String get gameOver =&gt; 'Fin del juego';

@override
String get gameOverDesc =&gt; 'Alcanzaste %s errores. Tiempo: %s';

@override
String get back =&gt; 'Atrás';
```

```
@override
String get retry =&gt; 'Reintentar';

@override
String get congrats =&gt; '¡Felicitaciones!';

@override
String get congratsDesc =&gt; '¡Completado en %s!';

@override
String get confirm =&gt; 'OK';

@override
String get nextLevel =&gt; 'Siguiente';

@override
String get newGame =&gt; 'Nueva partida';

@override
String get selectDifficulty =&gt; 'Seleccionar dificultad';

@override
String get difficultyEasy =&gt; 'Fácil';

@override
String get difficultyMedium =&gt; 'Medio';

@override
String get difficultyHard =&gt; 'Difícil';

@override
String get replayLevel =&gt; 'Repetir';

@override
String get difficulty =&gt; 'Dificultad';

@override
String get campaignTitle =&gt; 'Campaña';

@override
String get campaignSubtitle =&gt; '999 niveles, la dificultad aumenta cada 9 niveles';

@override
String get modeEasy =&gt; 'Fácil';

@override
String get modeMedium =&gt; 'Medio';

@override
```

```
  String get modeHard =&gt; 'Difícil';

  @override
  String get modeLevels =&gt; '999 niveles';

  @override
  String get startLevel =&gt; 'Iniciar nivel %s';

  @override
  String get unlockedTo =&gt; 'Desbloqueado hasta el nivel %s';

  @override
  String get clickToStart =&gt; 'Toca para comenzar';

  @override
  String get unlockedToClickToStart =&gt; 'Desbloqueado hasta el nivel %s. Toca para comenzar';
}
import 'app_localizations.dart';

class AppLocalizationsFr extends AppLocalizations {
  @override
  String get appTitle =&gt; 'CS Sudoku';

  @override
  String get sudoku =&gt; 'Sudoku';

  @override
  String get loading =&gt; 'Chargement...';

  @override
  String get slogan =&gt; 'Jeu classique de logique numérique';

  @override
  String get freeMode =&gt; 'Partie libre';

  @override
  String get campaignMode =&gt; 'Campagne';

  @override
  String get rulesTitle =&gt; 'Règles du Sudoku';

  @override
  String get rulesGoal =&gt; 'Objectif';

  @override
  String get rulesGoalDesc =&gt;
      'Remplissez la grille 9×9 avec les chiffres de 1 à 9 pour que chaque ligne, '
      'colonne et bloc 3×3 contienne chaque nombre exactement une fois.';

  @override
```

```
String get rulesBasic =&gt; 'Règles';

@override
String get rulesBasicItems =&gt;
    '• Pas de répétition dans chaque ligne\n'
    '• Pas de répétition dans chaque colonne\n'
    '• Pas de répétition dans chaque bloc 3×3\n'
    '• Les chiffres donnés ne peuvent pas être modifiés';

@override
String get rulesOps =&gt; 'Comment jouer';

@override
String get rulesOpsItems =&gt;
    '• Appuyez sur une cellule pour sélectionner, puis sur un chiffre pour remplir\n'
    '• Retour instantané : vert = correct, rouge = incorrect\n'
    '• Le mode campagne a une limite de 10 erreurs';

@override
String get rulesGotIt =&gt; 'Compris';

@override
String get levelFormat =&gt; 'Niveau %s';

@override
String get errorsFormat =&gt; 'Erreurs : %s/%s';

@override
String get gameOver =&gt; 'Fin de partie';

@override
String get gameOverDesc =&gt; 'Vous avez atteint %s erreurs. Temps : %s';

@override
String get back =&gt; 'Retour';

@override
String get retry =&gt; 'Réessayer';

@override
String get congrats =&gt; 'Félicitations !';

@override
String get congratsDesc =&gt; 'Terminé en %s !';

@override
String get confirm =&gt; 'OK';

@override
String get nextLevel =&gt; 'Suivant';
```

```
@override
String get newGame =&gt; 'Nouvelle partie';

@override
String get selectDifficulty =&gt; 'Choisir la difficulté';

@override
String get difficultyEasy =&gt; 'Facile';

@override
String get difficultyMedium =&gt; 'Moyen';

@override
String get difficultyHard =&gt; 'Difficile';

@override
String get replayLevel =&gt; 'Rejouer';

@override
String get difficulty =&gt; 'Difficulté';

@override
String get campaignTitle =&gt; 'Campagne';

@override
String get campaignSubtitle =&gt; '999 niveaux, difficulté croissante tous les 9 niveaux';

@override
String get modeEasy =&gt; 'Facile';

@override
String get modeMedium =&gt; 'Moyen';

@override
String get modeHard =&gt; 'Difficile';

@override
String get modeLevels =&gt; '999 niveaux';

@override
String get startLevel =&gt; 'Commencer le niveau %s';

@override
String get unlockedTo =&gt; 'Débloqué jusqu\'au niveau %s';

@override
String get clickToStart =&gt; 'Appuyez pour commencer';

@override
```

```
  String get unlockedToClickToStart =&gt; 'Débloqué jusqu\'au niveau %s. Appuyez pour commencer';
}
import 'app_localizations.dart';

class AppLocalizationsHi extends AppLocalizations {
  @override
  String get appTitle =&gt; 'CS सुडोकू';

  @override
  String get sudoku =&gt; 'सुडोकू';

  @override
  String get loading =&gt; 'लोड हो रहा है...';

  @override
  String get slogan =&gt; 'क्लासकि नंबर लॉजकि गेम';

  @override
  String get freeMode =&gt; 'फ्री प्ले';

  @override
  String get campaignMode =&gt; 'कैंपेन';

  @override
  String get rulesTitle =&gt; 'सुडोकू नयिम';

  @override
  String get rulesGoal =&gt; 'लक्ष्य';

  @override
  String get rulesGoalDesc =&gt;
    '9×9 ग्रडि को 1-9 अंकों से भरें ताकि प्रत्येक पंक्त,ि स्तंभ और '
    '3×3 बॉक्स में प्रत्येक संख्या बलि्कुल एक बार आए।';

  @override
  String get rulesBasic =&gt; 'नयिम';

  @override
  String get rulesBasicItems =&gt;
    '• प्रत्येक पंक्त में दोहराव नहीं\n'
    '• प्रत्येक स्तंभ में दोहराव नहीं\n'
    '• प्रत्येक 3×3 बॉक्स में दोहराव नहीं\n'
    '• दए गए अंक बदले नहीं जा सकते';

  @override
  String get rulesOps =&gt; 'कैसे खेलें';

  @override
  String get rulesOpsItems =&gt;
    '• सेल चुनने के लए टैप करें, फरि भरने के लए अंक टैप करें\n'
```

```
    '• तुरंत फीडबैक: हरा = सही, लाल = गलत\n'
    '• कैंपेन मोड में 10 गलतियों की सीमा';

@override
String get rulesGotIt =&gt; 'समझ गया';

@override
String get levelFormat =&gt; 'स्तर %s';

@override
String get errorsFormat =&gt; 'गलतियाँ: %s/%s';

@override
String get gameOver =&gt; 'गेम ओवर';

@override
String get gameOverDesc =&gt; 'आपने %s गलतियाँ पूरी कीं। समय: %s';

@override
String get back =&gt; 'वापस';

@override
String get retry =&gt; 'पुनः प्रयास';

@override
String get congrats =&gt; 'बधाई हो!';

@override
String get congratsDesc =&gt; '%s में पूर्ण!';

@override
String get confirm =&gt; 'ठीक है';

@override
String get nextLevel =&gt; 'अगला';

@override
String get newGame =&gt; 'नया गेम';

@override
String get selectDifficulty =&gt; 'कठिनाई चुनें';

@override
String get difficultyEasy =&gt; 'आसान';

@override
String get difficultyMedium =&gt; 'मध्यम';

@override
String get difficultyHard =&gt; 'कठिन';
```

```
  @override
  String get replayLevel =&gt; 'दोबारा खेलें';

  @override
  String get difficulty =&gt; 'कठिनाई';

  @override
  String get campaignTitle =&gt; 'कैंपेन';

  @override
  String get campaignSubtitle =&gt; '999 स्तर, हर 9 स्तर पर कठिनाई बढ़ती है';

  @override
  String get modeEasy =&gt; 'आसान';

  @override
  String get modeMedium =&gt; 'मध्यम';

  @override
  String get modeHard =&gt; 'कठिन';

  @override
  String get modeLevels =&gt; '999 स्तर';

  @override
  String get startLevel =&gt; 'स्तर %s शुरू करें';

  @override
  String get unlockedTo =&gt; 'स्तर %s तक अनलॉक';

  @override
  String get clickToStart =&gt; 'शुरू करने के लिए टैप करें';

  @override
  String get unlockedToClickToStart =&gt; 'स्तर %s तक अनलॉक। शुरू करने के लिए टैप करें';
}
import 'app_localizations.dart';

class AppLocalizationsId extends AppLocalizations {
  @override
  String get appTitle =&gt; 'CS Sudoku';

  @override
  String get sudoku =&gt; 'Sudoku';

  @override
  String get loading =&gt; 'Memuat...';

  @override
```

```
String get slogan =&gt; 'Permainan logika angka klasik';

@override
String get freeMode =&gt; 'Main bebas';

@override
String get campaignMode =&gt; 'Kampanye';

@override
String get rulesTitle =&gt; 'Aturan Sudoku';

@override
String get rulesGoal =&gt; 'Tujuan';

@override
String get rulesGoalDesc =&gt;
    'Isi grid 9×9 dengan angka 1–9 sehingga setiap baris, kolom, '
    'dan kotak 3×3 berisi setiap angka tepat satu kali.';

@override
String get rulesBasic =&gt; 'Aturan';

@override
String get rulesBasicItems =&gt;
    '• Tidak ada pengulangan di setiap baris\n'
    '• Tidak ada pengulangan di setiap kolom\n'
    '• Tidak ada pengulangan di setiap kotak 3×3\n'
    '• Angka yang diberikan tidak dapat diubah';

@override
String get rulesOps =&gt; 'Cara bermain';

@override
String get rulesOpsItems =&gt;
    '• Ketuk sel untuk memilih, lalu ketuk angka untuk mengisi\n'
    '• Umpan balik instan: hijau = benar, merah = salah\n'
    '• Mode kampanye memiliki batas 10 kesalahan';

@override
String get rulesGotIt =&gt; 'Mengerti';

@override
String get levelFormat =&gt; 'Level %s';

@override
String get errorsFormat =&gt; 'Kesalahan: %s/%s';

@override
String get gameOver =&gt; 'Game over';
```

```
@override
String get gameOverDesc =&gt; 'Anda mencapai %s kesalahan. Waktu: %s';

@override
String get back =&gt; 'Kembali';

@override
String get retry =&gt; 'Coba lagi';

@override
String get congrats =&gt; 'Selamat!';

@override
String get congratsDesc =&gt; 'Selesai dalam %s!';

@override
String get confirm =&gt; 'OK';

@override
String get nextLevel =&gt; 'Berikutnya';

@override
String get newGame =&gt; 'Permainan baru';

@override
String get selectDifficulty =&gt; 'Pilih kesulitan';

@override
String get difficultyEasy =&gt; 'Mudah';

@override
String get difficultyMedium =&gt; 'Sedang';

@override
String get difficultyHard =&gt; 'Sulit';

@override
String get replayLevel =&gt; 'Main ulang';

@override
String get difficulty =&gt; 'Kesulitan';

@override
String get campaignTitle =&gt; 'Kampanye';

@override
String get campaignSubtitle =&gt; '999 level, kesulitan meningkat setiap 9 level';

@override
String get modeEasy =&gt; 'Mudah';
```

```
  @override
  String get modeMedium =&gt; 'Sedang';

  @override
  String get modeHard =&gt; 'Sulit';

  @override
  String get modeLevels =&gt; '999 level';

  @override
  String get startLevel =&gt; 'Mulai level %s';

  @override
  String get unlockedTo =&gt; 'Terbuka hingga level %s';

  @override
  String get clickToStart =&gt; 'Ketuk untuk memulai';

  @override
  String get unlockedToClickToStart =&gt; 'Terbuka hingga level %s. Ketuk untuk memulai';
}
import 'app_localizations.dart';

class AppLocalizationsIt extends AppLocalizations {
  @override
  String get appTitle =&gt; 'CS Sudoku';

  @override
  String get sudoku =&gt; 'Sudoku';

  @override
  String get loading =&gt; 'Caricamento...';

  @override
  String get slogan =&gt; 'Gioco classico di logica numerica';

  @override
  String get freeMode =&gt; 'Gioco libero';

  @override
  String get campaignMode =&gt; 'Campagna';

  @override
  String get rulesTitle =&gt; 'Regole del Sudoku';

  @override
  String get rulesGoal =&gt; 'Obiettivo';

  @override
```

```
String get rulesGoalDesc =&gt;
    'Riempi la griglia 9×9 con le cifre da 1 a 9 in modo che ogni riga, '
    'colonna e riquadro 3×3 contenga ogni numero esattamente una volta.';

@override
String get rulesBasic =&gt; 'Regole';

@override
String get rulesBasicItems =&gt;
    '• Nessuna ripetizione in ogni riga\n'
    '• Nessuna ripetizione in ogni colonna\n'
    '• Nessuna ripetizione in ogni riquadro 3×3\n'
    '• I numeri dati non possono essere modificati';

@override
String get rulesOps =&gt; 'Come giocare';

@override
String get rulesOpsItems =&gt;
    '• Tocca una cella per selezionare, poi tocca un numero per riempire\n'
    '• Feedback immediato: verde = corretto, rosso = sbagliato\n'
    '• La modalità campagna ha un limite di 10 errori';

@override
String get rulesGotIt =&gt; 'Capito';

@override
String get levelFormat =&gt; 'Livello %s';

@override
String get errorsFormat =&gt; 'Errori: %s/%s';

@override
String get gameOver =&gt; 'Game over';

@override
String get gameOverDesc =&gt; 'Hai raggiunto %s errori. Tempo: %s';

@override
String get back =&gt; 'Indietro';

@override
String get retry =&gt; 'Riprova';

@override
String get congrats =&gt; 'Congratulazioni!';

@override
String get congratsDesc =&gt; 'Completato in %s!';
```

```
@override
String get confirm =&gt; 'OK';

@override
String get nextLevel =&gt; 'Avanti';

@override
String get newGame =&gt; 'Nuova partita';

@override
String get selectDifficulty =&gt; 'Seleziona difficoltà';

@override
String get difficultyEasy =&gt; 'Facile';

@override
String get difficultyMedium =&gt; 'Medio';

@override
String get difficultyHard =&gt; 'Difficile';

@override
String get replayLevel =&gt; 'Riprova';

@override
String get difficulty =&gt; 'Difficoltà';

@override
String get campaignTitle =&gt; 'Campagna';

@override
String get campaignSubtitle =&gt; '999 livelli, difficoltà aumenta ogni 9 livelli';

@override
String get modeEasy =&gt; 'Facile';

@override
String get modeMedium =&gt; 'Medio';

@override
String get modeHard =&gt; 'Difficile';

@override
String get modeLevels =&gt; '999 livelli';

@override
String get startLevel =&gt; 'Inizia livello %s';

@override
String get unlockedTo =&gt; 'Sbloccato fino al livello %s';
```

```
  @override
  String get clickToStart =&gt; 'Tocca per iniziare';

  @override
  String get unlockedToClickToStart =&gt; 'Sbloccato fino al livello %s. Tocca per iniziare';
}
import 'app_localizations.dart';

class AppLocalizationsJa extends AppLocalizations {
  @override
  String get appTitle =&gt; 'CS 数独';

  @override
  String get sudoku =&gt; '数独';

  @override
  String get loading =&gt; '読み込み中...';

  @override
  String get slogan =&gt; 'クラシック数字ロジックゲーム';

  @override
  String get freeMode =&gt; 'フリープレイ';

  @override
  String get campaignMode =&gt; 'キャンペーン';

  @override
  String get rulesTitle =&gt; '数独のルール';

  @override
  String get rulesGoal =&gt; '目標';

  @override
  String get rulesGoalDesc =&gt;
      '9×9のグリッドに1〜9の数字を入れ、各行・各列・各3×3のブロックに '
      '各数字が1回ずつ入るようにします。';

  @override
  String get rulesBasic =&gt; 'ルール';

  @override
  String get rulesBasicItems =&gt;
      '• 各行に重複なし\n'
      '• 各列に重複なし\n'
      '• 各3×3ブロックに重複なし\n'
      '• 最初から入っている数字は変更不可';

  @override
```

```
String get rulesOps =&gt; '遊び方';

@override
String get rulesOpsItems =&gt;
    '• セルをタップして選択し、数字をタップして入力\n'
    '• 即時フィードバック：緑 = 正解、赤 = 不正解\n'
    '• キャンペーンモードは10回までミス可能';

@override
String get rulesGotIt =&gt; 'わかりました';

@override
String get levelFormat =&gt; 'レベル %s';

@override
String get errorsFormat =&gt; 'ミス: %s/%s';

@override
String get gameOver =&gt; 'ゲームオーバー';

@override
String get gameOverDesc =&gt; '%s回ミスしました。タイム: %s';

@override
String get back =&gt; '戻る';

@override
String get retry =&gt; 'リトライ';

@override
String get congrats =&gt; 'おめでとうございます！';

@override
String get congratsDesc =&gt; '%sでクリア！';

@override
String get confirm =&gt; 'OK';

@override
String get nextLevel =&gt; '次へ';

@override
String get newGame =&gt; '新しいゲーム';

@override
String get selectDifficulty =&gt; '難易度を選択';

@override
String get difficultyEasy =&gt; '簡単';
```

```
  @override
  String get difficultyMedium =&gt; '普通';

  @override
  String get difficultyHard =&gt; '難しい';

  @override
  String get replayLevel =&gt; 'リプレイ';

  @override
  String get difficulty =&gt; '難易度';

  @override
  String get campaignTitle =&gt; 'キャンペーン';

  @override
  String get campaignSubtitle =&gt; '999レベル、9レベルごとに難易度上昇';

  @override
  String get modeEasy =&gt; '簡単';

  @override
  String get modeMedium =&gt; '普通';

  @override
  String get modeHard =&gt; '難しい';

  @override
  String get modeLevels =&gt; '999レベル';

  @override
  String get startLevel =&gt; 'レベル %s を開始';

  @override
  String get unlockedTo =&gt; 'レベル %s まで解放';

  @override
  String get clickToStart =&gt; 'タップして開始';

  @override
  String get unlockedToClickToStart =&gt; 'レベル %s まで解放。タップして開始';
}
import 'app_localizations.dart';

class AppLocalizationsKo extends AppLocalizations {
  @override
  String get appTitle =&gt; 'CS 스도쿠';

  @override
  String get sudoku =&gt; '스도쿠';
```

```
class _ModeCard extends StatelessWidget {
  const _ModeCard({
    required this.name,
    required this.modeLevels,
    required this.icon,
    required this.color,
    required this.onTap,
  });

  final String name;
  final String modeLevels;
  final IconData icon;
  final Color color;
  final VoidCallback onTap;

  @override
  Widget build(BuildContext context) {
    return Card(
      margin: const EdgeInsets.only(bottom: 16),
      elevation: 4,
      shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),
      child: InkWell(
        onTap: onTap,
        borderRadius: BorderRadius.circular(16),
        child: Padding(
          padding: const EdgeInsets.all(24),
          child: Row(
            children: [
              Container(
                padding: const EdgeInsets.all(16),
                decoration: BoxDecoration(
                  color: color.withValues(alpha: 0.2),
                  borderRadius: BorderRadius.circular(12),
                ),
                child: Icon(icon, size: 36, color: color),
              ),
              const SizedBox(width: 20),
              Expanded(
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Text(
                      name,
                      style: const TextStyle(
                        fontSize: 20,
                        fontWeight: FontWeight.bold,
                      ),
                    ),
                    const SizedBox(height: 4),
                    Text(
```

```
                    modeLevels,
                    style: TextStyle(
                      fontSize: 14,
                      color: Colors.grey.shade600,
                    ),
                  ),
                ],
              ),
            ),
            const Icon(Icons.chevron_right, color: Colors.grey),
          ],
        ),
      ),
    ),
  );
 }
}
import 'dart:async';

import 'package:flutter/material.dart';

import '../l10n/app_localizations.dart';
import '../services/campaign_progress.dart';
import '../sudoku/sudoku_generator.dart';
import '../sudoku/sudoku_validator.dart';
import '../widgets/number_pad.dart';
import '../widgets/sudoku_grid.dart';

class GameScreen extends StatefulWidget {
 const GameScreen({
   super.key,
   this.campaignMode,
   this.campaignLevel,
 });

 /// Campaign mode: 1=Easy, 2=Medium, 3=Hard
 final int? campaignMode;
 /// Campaign level: 1-999
 final int? campaignLevel;

 @override
 State&lt;GameScreen&gt; createState() =&gt; _GameScreenState();
}

class _GameScreenState extends State&lt;GameScreen&gt; {
 List&lt;List&lt;int&gt;&gt; _grid = List.generate(9, (_) =&gt; List.filled(9, 0));
 List&lt;List&lt;int&gt;&gt; _initialPuzzle = List.generate(9, (_) =&gt; List.filled(9, 0));
 List&lt;List&lt;int&gt;&gt; _solution = List.generate(9, (_) =&gt; List.filled(9, 0));
 ({int row, int col})? _selectedCell;
 bool _isNotesMode = false;
```

```
Set&lt;({int row, int col})&gt; _errorCells = {};
Set&lt;({int row, int col})&gt; _correctCells = {};
int _difficulty = 2; // 1=Easy, 2=Medium, 3=Hard (Free Play)
int _elapsedSeconds = 0;
Timer? _timer;
bool _isCompleted = false;

bool get _isCampaign =&gt; widget.campaignMode != null &amp;&amp; widget.campaignLevel != null;
bool _isLoading = true;
int _errorCount = 0;
static const int _maxErrors = 10; // Campaign error limit
bool _isGameOver = false;

@override
void initState() {
  super.initState();
  WidgetsBinding.instance.addPostFrameCallback((_) {
    if (!mounted) return;
    _newGame();
    if (mounted) {
      setState(() =&gt; _isLoading = false);
    }
  });
}

@override
void dispose() {
  _timer?.cancel();
  super.dispose();
}

void _newGame() {
  _timer?.cancel();
  _timer = null;
  _elapsedSeconds = 0;
  _isCompleted = false;
  _errorCount = 0;
  _isGameOver = false;

  final generator = SudokuGenerator();
  final result = _isCampaign
      ? generator.generateForLevel(
          mode: widget.campaignMode!,
          level: widget.campaignLevel!,
        )
      : generator.generatePuzzleWithSolution(difficulty: _difficulty);
  _initialPuzzle = result.puzzle;
  _solution = result.solution;
  _grid = _initialPuzzle.map((row) =&gt; List&lt;int&gt;.from(row)).toList();
  _selectedCell = null;
```

```
  _errorCells = {};
  _correctCells = {};

  _timer = Timer.periodic(const Duration(seconds: 1), (_) {
    if (!_isCompleted) {
      setState(() =&gt; _elapsedSeconds++);
    }
  });
}

void _onCellTap(int row, int col) {
  if (_initialPuzzle[row][col] != 0) return; // Fixed cells are not editable

  setState(() {
    _selectedCell = (row: row, col: col);
  });
}

void _onNumberTap(int number) {
  if (_selectedCell == null) return;
  if (_isCampaign &amp;&amp; _isGameOver) return;

  final (row: row, col: col) = _selectedCell!;
  if (_initialPuzzle[row][col] != 0) return;

  setState(() {
    if (_isNotesMode) {
      // Notes mode not implemented yet, fill number directly
      _grid[row][col] = number;
      _checkAndUpdateFeedback(row, col, number);
    } else {
      _grid[row][col] = number;
      _checkAndUpdateFeedback(row, col, number);

      if (SudokuValidator.isComplete(_grid)) {
        _isCompleted = true;
        _timer?.cancel();
        _showWinDialog();
      }
    }
  });
}

/// Check if input is correct against solution, update correct/error highlights
void _checkAndUpdateFeedback(int row, int col, int number) {
  final isCorrect = number == _solution[row][col];
  if (isCorrect) {
    _correctCells = {..._correctCells, (row: row, col: col)};
    _errorCells = _errorCells.difference({(row: row, col: col)});
  } else {
```

```
    _errorCells = {..._errorCells, (row: row, col: col)};
    _correctCells = _correctCells.difference({(row: row, col: col)});
    if (_isCampaign) {
      _errorCount++;
      if (_errorCount &gt;= _maxErrors) {
        _isGameOver = true;
        _timer?.cancel();
        _showGameOverDialog();
      }
    }
  }
}

void _onClear() {
  if (_selectedCell == null) return;
  if (_isCampaign &amp;&amp; _isGameOver) return;

  final (row: row, col: col) = _selectedCell!;
  if (_initialPuzzle[row][col] != 0) return;

  setState(() {
    _grid[row][col] = 0;
    _errorCells = _errorCells.difference({(row: row, col: col)});
    _correctCells = _correctCells.difference({(row: row, col: col)});
  });
}

void _onNotes() {
  setState(() =&gt; _isNotesMode = !_isNotesMode);
}

void _showGameOverDialog() {
  if (!mounted) return;
  final l10n = AppLocalizations.of(context);
  showDialog(
    context: context,
    barrierDismissible: false,
    builder: (ctx) =&gt; AlertDialog(
      title: Text(l10n.gameOver),
      content: Text(
        l10n.gameOverDescFormatted(_maxErrors, _formatTime(_elapsedSeconds)),
      ),
      actions: [
        TextButton(
          onPressed: () {
            Navigator.pop(ctx);
            Navigator.pop(context, true);
          },
          child: Text(l10n.back),
        ),
```

```
        FilledButton(
          onPressed: () {
            Navigator.pop(ctx);
            _newGame();
            setState(() {});
          },
          child: Text(l10n.retry),
        ),
      ],
    ),
  );
}

void _showWinDialog() async {
  if (_isCampaign) {
    await CampaignProgress.unlockNextLevel(
      widget.campaignMode!,
      widget.campaignLevel!,
    );
  }

  if (!mounted) return;
  final l10n = AppLocalizations.of(context);
  showDialog(
    context: context,
    barrierDismissible: false,
    builder: (ctx) =&gt; AlertDialog(
      title: Text(l10n.congrats),
      content: Text(
        l10n.congratsDescFormatted(_formatTime(_elapsedSeconds)),
      ),
      actions: [
        TextButton(
          onPressed: () {
            Navigator.pop(ctx);
            if (_isCampaign) {
              Navigator.pop(context, true);
            }
          },
          child: Text(l10n.confirm),
        ),
        if (_isCampaign &amp;&amp; widget.campaignLevel! &lt; 999)
          FilledButton(
            onPressed: () {
              Navigator.pop(ctx);
              Navigator.pop(context, widget.campaignLevel! + 1);
            },
            child: Text(l10n.nextLevel),
          )
        else if (!_isCampaign)
```

```
        FilledButton(
          onPressed: () {
            Navigator.pop(ctx);
            _newGame();
          },
          child: Text(l10n.newGame),
        ),
      ],
    ),
  );
}

String _formatTime(int seconds) {
  final m = seconds ~/ 60;
  final s = seconds % 60;
  return '${m.toString().padLeft(2, '0')}:${s.toString().padLeft(2, '0')}';
}

void _showDifficultyDialog() {
  final l10n = AppLocalizations.of(context);
  showDialog(
    context: context,
    builder: (ctx) =&gt; AlertDialog(
      title: Text(l10n.selectDifficulty),
      content: Column(
        mainAxisSize: MainAxisSize.min,
        children: [
          ListTile(
            title: Text(l10n.difficultyEasy),
            leading: Icon(
              _difficulty == 1 ? Icons.radio_button_checked : Icons.radio_button_off,
            ),
            onTap: () {
              _difficulty = 1;
              Navigator.pop(ctx);
              _newGame();
            },
          ),
          ListTile(
            title: Text(l10n.difficultyMedium),
            leading: Icon(
              _difficulty == 2 ? Icons.radio_button_checked : Icons.radio_button_off,
            ),
            onTap: () {
              _difficulty = 2;
              Navigator.pop(ctx);
              _newGame();
            },
          ),
          ListTile(
```

```
          title: Text(l10n.difficultyHard),
          leading: Icon(
            _difficulty == 3 ? Icons.radio_button_checked : Icons.radio_button_off,
          ),
          onTap: () {
            _difficulty = 3;
            Navigator.pop(ctx);
            _newGame();
          },
        ),
      ],
    ),
  ),
 );
}

@override
Widget build(BuildContext context) {
 return Scaffold(
  backgroundColor: const Color(0xFFE8F5E9),
  appBar: AppBar(
   title: Text(
    _isCampaign
      ? AppLocalizations.of(context).level(widget.campaignLevel!)
      : AppLocalizations.of(context).sudoku,
    style: const TextStyle(
     fontWeight: FontWeight.bold,
     fontSize: 22,
    ),
   ),
   backgroundColor: const Color(0xFF2E7D32),
   foregroundColor: Colors.white,
   elevation: 0,
   actions: [
    if (_isCampaign)
     Padding(
      padding: const EdgeInsets.only(right: 8),
      child: Center(
       child: Text(
        AppLocalizations.of(context).errors(_errorCount, _maxErrors),
        style: TextStyle(
         fontSize: 14,
         fontWeight: FontWeight.w600,
         color: _errorCount &gt;= _maxErrors - 2
           ? Colors.red.shade200
           : Colors.white,
        ),
       ),
      ),
     ),
```

```
    Padding(
      padding: const EdgeInsets.only(right: 12),
      child: Center(
        child: Text(
          _formatTime(_elapsedSeconds),
          style: const TextStyle(
            fontSize: 18,
            fontWeight: FontWeight.w600,
          ),
        ),
      ),
    ),
    IconButton(
      icon: const Icon(Icons.refresh),
      onPressed: _newGame,
      tooltip: _isCampaign
          ? AppLocalizations.of(context).replayLevel
          : AppLocalizations.of(context).newGame,
    ),
    if (!_isCampaign)
      IconButton(
        icon: const Icon(Icons.tune),
        onPressed: _showDifficultyDialog,
        tooltip: AppLocalizations.of(context).difficulty,
      ),
  ],
),
body: SafeArea(
  child: _isLoading
      ? Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              const CircularProgressIndicator(color: Color(0xFF2E7D32)),
              const SizedBox(height: 16),
              Text(
                AppLocalizations.of(context).loading,
                style: const TextStyle(fontSize: 16),
              ),
            ],
          ),
        )
      : LayoutBuilder(
    builder: (context, constraints) {
      return SingleChildScrollView(
        padding: const EdgeInsets.all(16),
        child: ConstrainedBox(
          constraints: BoxConstraints(
            minHeight: constraints.maxHeight - 32,
          ),
```

```
              child: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  const SizedBox(height: 16),
                  SudokuGrid(
                    grid: _grid,
                    initialPuzzle: _initialPuzzle,
                    selectedCell: _selectedCell,
                    onCellTap: _onCellTap,
                    errorCells: _errorCells,
                    correctCells: _correctCells,
                  ),
                  const SizedBox(height: 24),
                  NumberPad(
                    onNumberTap: _onNumberTap,
                    onClear: _onClear,
                    onNotes: _onNotes,
                    isNotesMode: _isNotesMode,
                    enabled: !(_isCampaign &amp;amp;&amp;amp; _isGameOver),
                  ),
                ],
              ),
            ),
          );
        },
      ),
    ),
  );
 }
}
import 'package:flutter/material.dart';

import '../l10n/app_localizations.dart';
import 'campaign_mode_screen.dart';
import 'game_screen.dart';

class HomeScreen extends StatelessWidget {
 const HomeScreen({super.key});

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   body: Container(
    decoration: const BoxDecoration(
     gradient: LinearGradient(
      begin: Alignment.topLeft,
      end: Alignment.bottomRight,
      colors: [
       Color(0xFF1B5E20),
       Color(0xFF2E7D32),
```

```
      Color(0xFF388E3C),
    ],
   ),
  ),
 ),
 child: SafeArea(
  child: Center(
   child: Padding(
    padding: const EdgeInsets.all(32),
    child: Column(
     mainAxisAlignment: MainAxisAlignment.center,
     children: [
      Container(
       padding: const EdgeInsets.all(24),
       decoration: BoxDecoration(
        color: Colors.white.withValues(alpha: 0.2),
        borderRadius: BorderRadius.circular(24),
        boxShadow: [
         BoxShadow(
          color: Colors.black.withValues(alpha: 0.2),
          blurRadius: 20,
          offset: const Offset(0, 8),
         ),
        ],
       ),
       child: LayoutBuilder(
        builder: (context, constraints) {
         return SizedBox(
          width: constraints.maxWidth,
          child: FittedBox(
           fit: BoxFit.scaleDown,
           alignment: Alignment.center,
           child: Text(
            AppLocalizations.of(context).sudoku,
            style: const TextStyle(
             fontSize: 72,
             fontWeight: FontWeight.bold,
             color: Colors.white,
             letterSpacing: 8,
            ),
            maxLines: 1,
           ),
          ),
         );
        },
       ),
      ),
      const SizedBox(height: 48),
      Text(
       AppLocalizations.of(context).slogan,
       style: TextStyle(
```

```
    fontSize: 18,
    color: Colors.white70,
    letterSpacing: 2,
  ),
),
const SizedBox(height: 48),
SizedBox(
  width: double.infinity,
  height: 56,
  child: FilledButton(
    onPressed: () {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) =&gt; const GameScreen(),
        ),
      );
    },
    style: FilledButton.styleFrom(
      backgroundColor: Colors.white,
      foregroundColor: const Color(0xFF2E7D32),
      elevation: 4,
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(16),
      ),
      textStyle: const TextStyle(
        fontSize: 20,
        fontWeight: FontWeight.bold,
      ),
    ),
    child: Text(AppLocalizations.of(context).freeMode),
  ),
),
const SizedBox(height: 16),
SizedBox(
  width: double.infinity,
  height: 56,
  child: OutlinedButton(
    onPressed: () {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) =&gt; const CampaignModeScreen(),
        ),
      );
    },
    style: OutlinedButton.styleFrom(
      foregroundColor: Colors.white,
      side: const BorderSide(color: Colors.white, width: 2),
      shape: RoundedRectangleBorder(
```

```
                  borderRadius: BorderRadius.circular(16),
                ),
                textStyle: const TextStyle(
                  fontSize: 20,
                  fontWeight: FontWeight.bold,
                ),
              ),
              child: Text(AppLocalizations.of(context).campaignMode),
            ),
          ),
          const SizedBox(height: 24),
          TextButton.icon(
            onPressed: () =&gt; _showRulesDialog(context),
            icon: const Icon(Icons.help_outline, color: Colors.white70, size: 20),
            label: Text(
              AppLocalizations.of(context).rulesTitle,
              style: TextStyle(
                color: Colors.white70,
                fontSize: 16,
              ),
            ),
          ),
        ],
      ),
    ),
  ),
 );
}

void _showRulesDialog(BuildContext context) {
 final l10n = AppLocalizations.of(context);
 showDialog(
  context: context,
  builder: (ctx) =&gt; AlertDialog(
   title: Text(l10n.rulesTitle),
   content: SingleChildScrollView(
    child: Column(
     mainAxisSize: MainAxisSize.min,
     crossAxisAlignment: CrossAxisAlignment.start,
     children: [
      Text(
       l10n.rulesGoal,
       style: const TextStyle(
        fontWeight: FontWeight.bold,
        fontSize: 16,
       ),
      ),
      const SizedBox(height: 8),
```

```
              Text(
                l10n.rulesGoalDesc,
                style: const TextStyle(height: 1.5),
              ),
              const SizedBox(height: 20),
              Text(
                l10n.rulesBasic,
                style: const TextStyle(
                  fontWeight: FontWeight.bold,
                  fontSize: 16,
                ),
              ),
              const SizedBox(height: 8),
              Text(
                l10n.rulesBasicItems,
                style: const TextStyle(height: 1.6),
              ),
              const SizedBox(height: 20),
              Text(
                l10n.rulesOps,
                style: const TextStyle(
                  fontWeight: FontWeight.bold,
                  fontSize: 16,
                ),
              ),
              const SizedBox(height: 8),
              Text(
                l10n.rulesOpsItems,
                style: const TextStyle(height: 1.6),
              ),
            ],
          ),
        ),
        actions: [
          FilledButton(
            onPressed: () =&gt; Navigator.pop(ctx),
            child: Text(l10n.rulesGotIt),
          ),
        ],
      ),
    );
  }
}
import 'package:flutter/material.dart';

import '../l10n/app_localizations.dart';
import '../services/campaign_progress.dart';
import 'game_screen.dart';

/// Level select - shows levels 1-999, can jump to a specific level
```

```
class LevelSelectScreen extends StatefulWidget {
  const LevelSelectScreen({
    super.key,
    required this.mode,
    required this.modeName,
  });

  final int mode;
  final String modeName;

  @override
  State&lt;LevelSelectScreen&gt; createState() =&gt; _LevelSelectScreenState();
}

class _LevelSelectScreenState extends State&lt;LevelSelectScreen&gt; {
  int _currentLevel = 1;

  @override
  void initState() {
    super.initState();
    _loadProgress();
  }

  Future&lt;void&gt; _loadProgress() async {
    final level = await CampaignProgress.getCurrentLevelAsync(widget.mode);
    setState(() =&gt; _currentLevel = level);
  }

  void _playLevel(int level) {
    if (level &gt; _currentLevel) return;
    if (!mounted) return;

    _navigateAndPlay(level);
  }

  Future&lt;void&gt; _navigateAndPlay(int level) async {
    if (!mounted) return;

    final navigator = Navigator.of(context);
    final result = await navigator.push&lt;Object&gt;(
      MaterialPageRoute(
        builder: (context) =&gt; GameScreen(
          campaignMode: widget.mode,
          campaignLevel: level,
        ),
      ),
    );

    if (!mounted) return;
    if (result != null) {
```

```
    await _loadProgress();
    if (result is int &amp;amp;&amp;amp; result &amp;lt;= 999) {
      _playLevel(result);
    }
  }
}

@override
Widget build(BuildContext context) {
  final l10n = AppLocalizations.of(context);
  return Scaffold(
    body: Container(
      decoration: const BoxDecoration(
        gradient: LinearGradient(
          begin: Alignment.topLeft,
          end: Alignment.bottomRight,
          colors: [
            Color(0xFF1B5E20),
            Color(0xFF2E7D32),
            Color(0xFF388E3C),
          ],
        ),
      ),
      child: SafeArea(
        child: Column(
          children: [
            AppBar(
              title: Text(
                '${widget.modeName} · ${l10n.level(_currentLevel)}',
                style: const TextStyle(
                  color: Colors.white,
                  fontWeight: FontWeight.bold,
                  fontSize: 18,
                ),
              ),
              backgroundColor: Colors.transparent,
              elevation: 0,
              leading: IconButton(
                icon: const Icon(Icons.arrow_back, color: Colors.white),
                onPressed: () =&amp;gt; Navigator.pop(context),
              ),
            ),
            Padding(
              padding: const EdgeInsets.symmetric(horizontal: 24),
              child: Text(
                l10n.unlockedToClickToStartFormatted(_currentLevel),
                style: const TextStyle(color: Colors.white70, fontSize: 14),
              ),
            ),
            const SizedBox(height: 12),
```

```
Padding(
  padding: const EdgeInsets.symmetric(horizontal: 24),
  child: SizedBox(
    width: double.infinity,
    height: 48,
    child: FilledButton.icon(
      onPressed: () =&gt; _playLevel(_currentLevel),
      icon: const Icon(Icons.play_arrow, size: 24),
      label: Text(l10n.startLevelFormatted(_currentLevel)),
      style: FilledButton.styleFrom(
        backgroundColor: Colors.white,
        foregroundColor: const Color(0xFF2E7D32),
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(12),
        ),
      ),
    ),
  ),
),
const SizedBox(height: 16),
Expanded(
  child: Container(
    margin: const EdgeInsets.symmetric(horizontal: 16),
    decoration: BoxDecoration(
      color: Colors.white,
      borderRadius: BorderRadius.circular(16),
      boxShadow: [
        BoxShadow(
          color: Colors.black.withValues(alpha: 0.2),
          blurRadius: 12,
          offset: const Offset(0, 4),
        ),
      ],
    ),
    child: ClipRRect(
      borderRadius: BorderRadius.circular(16),
      child: ListView.builder(
        padding: const EdgeInsets.all(16),
        itemCount: 167,
        itemBuilder: (context, rowIndex) {
          final startLevel = rowIndex * 6 + 1;
          return Padding(
            padding: const EdgeInsets.only(bottom: 8),
            child: Row(
              children: List.generate(6, (colIndex) {
                final level = startLevel + colIndex;
                if (level &gt; 999) return const SizedBox.shrink();
                final isCurrent = level == _currentLevel;
                final isUnlocked = level &lt;= _currentLevel;
                return Expanded(
```

```
                        child: Padding(
                          padding: EdgeInsets.only(
                            right: colIndex &lt; 5 ? 8 : 0,
                          ),
                          child: _LevelTile(
                            level: level,
                            isUnlocked: isUnlocked,
                            isCurrent: isCurrent,
                            onTap: () =&gt; _playLevel(level),
                          ),
                        ),
                      );
                    }),
                  ),
                );
              },
            ),
          ),
        ),
      ),
    ],
  ),
 ),
 ),
 );
 }
}

class _LevelTile extends StatelessWidget {
  const _LevelTile({
    required this.level,
    required this.isUnlocked,
    required this.isCurrent,
    required this.onTap,
  });

  final int level;
  final bool isUnlocked;
  final bool isCurrent;
  final VoidCallback onTap;

  @override
  Widget build(BuildContext context) {
    return SizedBox(
      height: 52,
      child: Material(
        color: isUnlocked
            ? (isCurrent ? const Color(0xFF2E7D32) : Colors.green.shade50)
            : Colors.grey.shade200,
        borderRadius: BorderRadius.circular(8),
```

```
        child: InkWell(
          onTap: isUnlocked ? onTap : null,
          borderRadius: BorderRadius.circular(8),
          child: Center(
            child: Text(
              '$level',
              style: TextStyle(
                fontSize: 16,
                fontWeight: isCurrent ? FontWeight.bold : FontWeight.w500,
                color: isUnlocked
                    ? (isCurrent ? Colors.white : Colors.green.shade800)
                    : Colors.grey.shade500,
              ),
            ),
          ),
        ),
      ),
    );
  }
}
import 'package:get_storage/get_storage.dart';

/// Campaign progress management
/// Uses get_storage for persistence; falls back to in-memory if plugin unavailable
/// 3 modes: Easy(1), Medium(2), Hard(3), 999 levels each
class CampaignProgress {
  static const _boxName = 'campaign_progress';
  static const _keyPrefix = 'campaign_level_';

  static GetStorage? _box;
  static bool _initialized = false;
  static bool _useMemoryFallback = false;
  static final Map&lt;int, int&gt; _memoryStore = {1: 1, 2: 1, 3: 1};

  static String _key(int mode) =&gt; '$_keyPrefix$mode';

  static Future&lt;void&gt; _tryInit() async {
    if (_initialized) return;
    try {
      await GetStorage.init(_boxName);
      _box = GetStorage(_boxName);
      _initialized = true;
    } catch (_) {
      _useMemoryFallback = true;
      _initialized = true;
    }
  }

  /// Initialize storage (non-blocking, fallback to memory on failure)
  static Future&lt;void&gt; ensureInitialized() async {
```

```
    await _tryInit();
  }

  static int _getFromMemory(int mode) =&gt; _memoryStore[mode] ?? 1;

  static void _setToMemory(int mode, int value) {
    _memoryStore[mode] = value;
  }

  /// Get current level for mode (1-999, default 1)
  static Future&lt;int&gt; getCurrentLevelAsync(int mode) async {
    await _tryInit();
    if (_useMemoryFallback) return _getFromMemory(mode);
    final v = _box!.read(_key(mode));
    return (v is int) ? v : 1;
  }

  /// Save progress (call after completing a level to unlock next)
  static Future&lt;void&gt; unlockNextLevel(int mode, int completedLevel) async {
    await _tryInit();
    final next = completedLevel + 1;
    if (next &gt; 999) return;

    _setToMemory(mode, next);
    if (!_useMemoryFallback) {
      try {
        await _box!.write(_key(mode), next);
      } catch (_) {
        _useMemoryFallback = true;
      }
    }
  }

  /// Check if a level is unlocked
  static Future&lt;bool&gt; isLevelUnlocked(int mode, int level) async {
    final current = await getCurrentLevelAsync(mode);
    return level &lt;= current;
  }
}
import 'dart:math';

/// Sudoku generator - generates solvable sudoku puzzles
class SudokuGenerator {
  final Random _random = Random();

  /// Generate a complete valid sudoku solution
  List&lt;List&lt;int&gt;&gt; generateSolution() {
    final grid = List.generate(9, (_) =&gt; List.filled(9, 0));

    _fillDiagonalBoxes(grid);
```

```
  _solve(grid);

  return grid;
}

/// Fill diagonal 3x3 boxes first (they are independent)
void _fillDiagonalBoxes(List&lt;List&lt;int&gt;&gt; grid) {
  for (int box = 0; box &lt; 9; box += 3) {
    _fillBox(grid, box, box);
  }
}

void _fillBox(List&lt;List&lt;int&gt;&gt; grid, int row, int col) {
  final numbers = List.generate(9, (i) =&gt; i + 1)..shuffle(_random);

  for (int i = 0; i &lt; 3; i++) {
    for (int j = 0; j &lt; 3; j++) {
      grid[row + i][col + j] = numbers[i * 3 + j];
    }
  }
}

/// Solve sudoku using backtracking
bool _solve(List&lt;List&lt;int&gt;&gt; grid) {
  for (int row = 0; row &lt; 9; row++) {
    for (int col = 0; col &lt; 9; col++) {
      if (grid[row][col] == 0) {
        final numbers = List.generate(9, (i) =&gt; i + 1)..shuffle(_random);

        for (final num in numbers) {
          if (_isValid(grid, row, col, num)) {
            grid[row][col] = num;

            if (_solve(grid)) {
              return true;
            }

            grid[row][col] = 0;
          }
        }
        return false;
      }
    }
  }
  return true;
}

bool _isValid(List&lt;List&lt;int&gt;&gt; grid, int row, int col, int num) {
  // Check row
  for (int x = 0; x &lt; 9; x++) {
```

```
    if (grid[row][x] == num) return false;
  }

  // Check column
  for (int x = 0; x &lt; 9; x++) {
    if (grid[x][col] == num) return false;
  }

  // Check 3x3 box
  final startRow = row - row % 3;
  final startCol = col - col % 3;
  for (int i = 0; i &lt; 3; i++) {
    for (int j = 0; j &lt; 3; j++) {
      if (grid[startRow + i][startCol + j] == num) return false;
    }
  }

  return true;
}

/// Generate sudoku puzzle and solution
/// difficulty: 1=Easy, 2=Medium, 3=Hard (for Free Play)
/// cellsToRemove: directly specify cells to remove (for Campaign)
/// More cells removed = harder puzzle (max 80, keep at least 1)
({List&lt;List&lt;int&gt;&gt; puzzle, List&lt;List&lt;int&gt;&gt; solution}) generatePuzzleWithSolution({
  int? difficulty,
  int? cellsToRemove,
}) {
  final solution = generateSolution();
  final puzzle = solution.map((row) =&gt; List&lt;int&gt;.from(row)).toList();

  final toRemove = (cellsToRemove ??
      (switch (difficulty ?? 2) {
        1 =&gt; 32 + _random.nextInt(7), // Easy: remove 32-38 cells
        3 =&gt; 55 + _random.nextInt(8), // Hard: remove 55-62 cells
        _ =&gt; 45 + _random.nextInt(8), // Medium: remove 45-52 cells
      }))
      .clamp(20, 80);

  int removed = 0;
  int attempts = 0;
  final maxAttempts = (toRemove * 5).clamp(200, 800);

  while (removed &lt; toRemove &amp;&amp; attempts &lt; maxAttempts) {
    final row = _random.nextInt(9);
    final col = _random.nextInt(9);

    if (puzzle[row][col] != 0) {
      final backup = puzzle[row][col];
      puzzle[row][col] = 0;
```

```
      // Ensure puzzle still has unique solution (simplified: only check solvable)
      if (_hasUniqueSolution(puzzle)) {
        removed++;
      } else {
        puzzle[row][col] = backup;
      }
    }
    attempts++;
  }

  return (puzzle: puzzle, solution: solution);
}

/// Campaign mode: generate puzzle by mode and level
/// mode: 1=Easy, 2=Medium, 3=Hard
/// level: 1-999, difficulty increases every 9 levels
({List&lt;List&lt;int&gt;&gt; puzzle, List&lt;List&lt;int&gt;&gt; solution}) generateForLevel({
  required int mode,
  required int level,
}) {
  // Every 9 levels form a tier, 111 tiers total (tier 0~110)
  final tier = (level - 1) ~/ 9;
  final tierFactor = tier / 111; // 0.0 ~ 0.99

  // Base and max cells to remove per mode (level 1 vs level 999)
  final (base, maxRemoved) = switch (mode) {
    1 =&gt; (28, 42), // Easy
    3 =&gt; (54, 70), // Hard
    _ =&gt; (42, 58), // Medium
  };

  final cellsToRemove = (base +
          (tierFactor * (maxRemoved - base)).round() +
          _random.nextInt(4))
      .clamp(20, 80);

  return generatePuzzleWithSolution(cellsToRemove: cellsToRemove);
}

/// Generate puzzle (legacy API)
List&lt;List&lt;int&gt;&gt; generatePuzzle({int difficulty = 2}) {
  return generatePuzzleWithSolution(difficulty: difficulty).puzzle;
}

/// Check if puzzle has unique solution (simplified: only check solvable)
bool _hasUniqueSolution(List&lt;List&lt;int&gt;&gt; grid) {
  final copy = grid.map((row) =&gt; List&lt;int&gt;.from(row)).toList();
  return _solve(copy);
}
```

```
}
/// Sudoku validator - checks if placement is valid
class SudokuValidator {
  /// Check if placing a number at the given position is valid
  static bool isValidPlacement(
    List&lt;List&lt;int&gt;&gt; grid,
    int row,
    int col,
    int num,
  ) {
    if (num &lt; 1 || num &gt; 9) return false;

    // Check row
    for (int x = 0; x &lt; 9; x++) {
      if (x != col &amp;&amp; grid[row][x] == num) return false;
    }

    // Check column
    for (int x = 0; x &lt; 9; x++) {
      if (x != row &amp;&amp; grid[x][col] == num) return false;
    }

    // Check 3x3 box
    final startRow = row - row % 3;
    final startCol = col - col % 3;
    for (int i = 0; i &lt; 3; i++) {
      for (int j = 0; j &lt; 3; j++) {
        final r = startRow + i;
        final c = startCol + j;
        if ((r != row || c != col) &amp;&amp; grid[r][c] == num) return false;
      }
    }

    return true;
  }

  /// Check if the entire sudoku is correctly completed
  static bool isComplete(List&lt;List&lt;int&gt;&gt; grid) {
    for (int row = 0; row &lt; 9; row++) {
      for (int col = 0; col &lt; 9; col++) {
        if (grid[row][col] == 0) return false;
        if (!isValidPlacement(grid, row, col, grid[row][col])) return false;
      }
    }
    return true;
  }
}
import 'package:flutter/material.dart';

class NumberPad extends StatelessWidget {
```

```
const NumberPad({
  super.key,
  required this.onNumberTap,
  required this.onClear,
  required this.onNotes,
  this.isNotesMode = false,
  this.enabled = true,
});

final void Function(int number) onNumberTap;
final VoidCallback onClear;
final VoidCallback onNotes;
final bool isNotesMode;
final bool enabled;

@override
Widget build(BuildContext context) {
  return LayoutBuilder(
    builder: (context, constraints) {
      // Calculate button size from available width to avoid overflow
      final availableWidth = constraints.maxWidth - 32; // minus padding
      final buttonSize = (availableWidth / 9).clamp(32.0, 56.0);
      final actionButtonSize = (buttonSize * 0.8).clamp(28.0, 48.0);

      return Container(
        padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 12),
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
              children: [
                for (int i = 1; i &lt;= 9; i++)
                  _PadButton(
                    label: i.toString(),
                    size: buttonSize,
                    onTap: enabled ? () =&gt; onNumberTap(i) : null,
                  ),
              ],
            ),
            const SizedBox(height: 12),
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                _PadButton(
                  label: '✎',
                  size: actionButtonSize,
                  onTap: enabled ? onNotes : null,
                  isHighlighted: isNotesMode,
                ),
```

```
                const SizedBox(width: 16),
                _PadButton(
                  label: '⌫',
                  size: actionButtonSize,
                  onTap: enabled ? onClear : null,
                ),
              ],
            ),
          ],
        ),
      );
    },
  );
 }
}

class _PadButton extends StatelessWidget {
 const _PadButton({
  required this.label,
  required this.size,
  required this.onTap,
  this.isHighlighted = false,
 });

 final String label;
 final double size;
 final VoidCallback? onTap;
 final bool isHighlighted;

 @override
 Widget build(BuildContext context) {
  return Opacity(
   opacity: onTap != null ? 1 : 0.5,
   child: Material(
    color: isHighlighted ? Colors.blue.shade100 : Colors.grey.shade100,
    borderRadius: BorderRadius.circular(12),
    child: InkWell(
     onTap: onTap,
     borderRadius: BorderRadius.circular(12),
     child: SizedBox(
      width: size,
      height: size,
      child: Center(
       child: Text(
        label,
        style: TextStyle(
         fontSize: size * 0.4,
         fontWeight: FontWeight.w600,
         color: isHighlighted ? Colors.blue.shade700 : Colors.black87,
        ),
```

```
            ),
          ),
        ),
      ),
    ),
  );
 }
}
import 'package:flutter/material.dart';

/// Sudoku grid layout constants
abstract final class _GridConstants {
 static const double minCellSize = 28.0;
 static const double maxCellSize = 42.0;
 static const double minFontSize = 14.0;
 static const double maxFontSize = 22.0;
 static const double minHintFontSize = 10.0;
 static const double maxHintFontSize = 14.0;
 static const double gridPadding = 16.0;
 static const double separatorSpace = 30.0;
 static const int gridSize = 9;
 static const int blockSize = 3;
}

class SudokuGrid extends StatelessWidget {
 const SudokuGrid({
   super.key,
   required this.grid,
   required this.initialPuzzle,
   required this.selectedCell,
   required this.onCellTap,
   this.errorCells = const {},
   this.correctCells = const {},
 });

 final List&lt;List&lt;int&gt;&gt; grid;
 final List&lt;List&lt;int&gt;&gt; initialPuzzle;
 final ({int row, int col})? selectedCell;
 final void Function(int row, int col) onCellTap;
 final Set&lt;({int row, int col})&gt; errorCells;
 final Set&lt;({int row, int col})&gt; correctCells;

 @override
 Widget build(BuildContext context) {
   return LayoutBuilder(
     builder: (context, constraints) {
       final maxW = constraints.maxWidth.isInfinite ? 400.0 : constraints.maxWidth;
       final availableWidth = maxW - _GridConstants.gridPadding;
       final cellSize = ((availableWidth - _GridConstants.separatorSpace) / _GridConstants.gridSize)
           .clamp(_GridConstants.minCellSize, _GridConstants.maxCellSize);
```

```
  final fontSize = (cellSize * 0.5).clamp(_GridConstants.minFontSize, _GridConstants.maxFontSize);
  final hintFontSize = (cellSize * 0.35).clamp(_GridConstants.minHintFontSize, _GridConstants.maxHintFontSize);

  return Container(
    padding: const EdgeInsets.all(8),
    decoration: BoxDecoration(
      color: Colors.white,
      borderRadius: BorderRadius.circular(12),
      boxShadow: [
        BoxShadow(
          color: Colors.black.withValues(alpha: 0.1),
          blurRadius: 12,
          offset: const Offset(0, 4),
        ),
      ],
    ),
    child: Column(
      mainAxisSize: MainAxisSize.min,
      children: [
        for (int row = 0; row &lt; _GridConstants.gridSize; row++) ...[
          if (row &gt; 0 &amp;&amp; row % _GridConstants.blockSize == 0)
            const _GridSeparator(isHorizontal: true),
          Row(
            mainAxisSize: MainAxisSize.min,
            children: [
              for (int col = 0; col &lt; _GridConstants.gridSize; col++) ...[
                if (col &gt; 0 &amp;&amp; col % _GridConstants.blockSize == 0)
                  _GridSeparator(isHorizontal: false, height: cellSize),
                RepaintBoundary(
                  child: _SudokuCell(
                    key: ValueKey('cell-$row-$col'),
                    value: grid[row][col],
                    fontSize: fontSize,
                    hintFontSize: hintFontSize,
                    cellSize: cellSize,
                    isFixed: initialPuzzle[row][col] != 0,
                    isSelected: selectedCell?.row == row &amp;&amp; selectedCell?.col == col,
                    hasError: errorCells.contains((row: row, col: col)),
                    isCorrect: correctCells.contains((row: row, col: col)),
                    onTap: () =&gt; onCellTap(row, col),
                  ),
                ),
              ],
            ],
          ),
        ],
      ],
    ),
  );
},
```

```
    );
  }
}

class _GridSeparator extends StatelessWidget {
  const _GridSeparator({
    required this.isHorizontal,
    this.height,
  });

  final bool isHorizontal;
  final double? height;

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: EdgeInsets.symmetric(
        vertical: isHorizontal ? 2 : 0,
        horizontal: isHorizontal ? 0 : 2,
      ),
      child: Container(
        width: isHorizontal ? null : 2,
        height: isHorizontal ? 2 : height,
        color: Colors.grey.shade400,
      ),
    );
  }
}

class _SudokuCell extends StatelessWidget {
  const _SudokuCell({
    super.key,
    required this.value,
    required this.fontSize,
    required this.hintFontSize,
    required this.cellSize,
    required this.isFixed,
    required this.isSelected,
    required this.hasError,
    required this.isCorrect,
    required this.onTap,
  });

  final int value;
  final double fontSize;
  final double hintFontSize;
  final double cellSize;
  final bool isFixed;
  final bool isSelected;
  final bool hasError;
```

```
  final bool isCorrect;
  final VoidCallback onTap;

  static const _animationDuration = Duration(milliseconds: 150);
  static const _cellMargin = EdgeInsets.all(1);
  static const _borderRadius = 4.0;

  bool get _isHighlighted =&gt; hasError || isCorrect || isSelected;

  (Color fill, Color border, Color text) get _colors {
    if (hasError) return (Colors.red.shade100, Colors.red, Colors.red.shade800);
    if (isCorrect) return (Colors.green.shade100, Colors.green, Colors.green.shade800);
    if (isSelected) return (Colors.blue.shade100, Colors.blue, Colors.blue.shade700);
    return (Colors.transparent, Colors.grey.shade300, Colors.black87);
  }

  @override
  Widget build(BuildContext context) {
    final (fill, border, text) = _colors;
    return GestureDetector(
      onTap: onTap,
      child: AnimatedContainer(
        duration: _animationDuration,
        width: cellSize,
        height: cellSize,
        margin: _cellMargin,
        decoration: BoxDecoration(
          color: fill,
          border: Border.all(
            color: border,
            width: _isHighlighted ? 2 : 0.5,
          ),
          borderRadius: BorderRadius.circular(_borderRadius),
        ),
        child: Center(
          child: value &gt; 0
              ? Text(
                  value.toString(),
                  style: TextStyle(
                    fontSize: fontSize,
                    fontWeight: isFixed ? FontWeight.bold : FontWeight.w500,
                    color: isFixed ? Colors.black87 : text,
                  ),
                )
              : null,
        ),
      ),
    );
  }
}
```