/\*

  Catchphrase (No Categories)

  Hardware:

    - Arduino Uno

    - 1602 LCD (parallel) on pins RS=8, E=9, D4=A0(14), D5=A1(15), D6=A2(16), D7=A3(17)

    - Backlight control on A4 (18). 220 ohm resiter between this pin (A4) and LCD pin A

    - SD TF card reader (CS=10, MOSI=11, MISO=12, SCK=13)

    - piezo Buzzer on D7

    - Buttons:

        START/STOP = D2

        TEAM1      = D3

        TEAM2      = D4

        NEXT       = D5

        CATEGORY   = D6  (repurposed as MUTE toggle)

\*/

//text file (notepad) must be called words.txt

//14 characters per LCD line. The formatter tries to split on a space so the clue can use up to 2 lines.

//if a word is longer than 13 characters it gets skipped (no words longer than 13 characters)

// Blank lines or lines starting with # are ignored.

// Format a clue to fit: TOP=14 chars, BOTTOM=16 chars.

// Returns a single String with length TOP\_TEXT\_LEN + BOTTOM\_TEXT\_LEN.

// First 14 chars -> top text window; last 16 chars -> bottom line.

// splits a phrase at a space if total phrase is longer than the top\_text length (14).

// if single word is longer than 14 letters, but no longer than 16, it is displayed on bottom line and top line is left empty (except the scores)

// any words 17+ letters is skipped.

// this is my script that works great. Just limited to 120 words per sd card

/\*

  Catchphrase (No Categories)

  - Keeps one 120-word deck across multiple rounds (no repeats within deck)

  - Builds the deck on FIRST Start press (not in setup) so the first batch is random each power-up

  - If words.txt has <=120 usable lines: uses all of them; reshuffles only after all shown once

  - If words.txt has >120 usable: random 120-sample; rebuilds a new random 120 only when deck is exhausted

\*/

#include <SPI.h>

#include <SD.h>

#include <LiquidCrystal.h>

// ===== Pins =====

const byte TRANSISTOR\_POWER\_PIN = 19; // A5

const byte START\_STOP\_PIN = 2;

const byte TEAM1\_PIN      = 3;

const byte TEAM2\_PIN      = 4;

const byte NEXT\_PIN       = 5;

const byte CATEGORY\_PIN   = 6;  // Mute toggle

const byte SPEAKER\_PIN    = 7;

const byte LCD\_PIN\_RS = 8;

const byte LCD\_PIN\_E  = 9;

const byte SD\_PIN\_CS  = 10;

const byte LCD\_PIN\_D4 = 14; // A0

const byte LCD\_PIN\_D5 = 15; // A1

const byte LCD\_PIN\_D6 = 16; // A2

const byte LCD\_PIN\_D7 = 17; // A3

const byte LCD\_PIN\_BL = 18; // A4 (backlight via 220Ω to LCD A)

// ===== LCD + SD =====

LiquidCrystal lcd(LCD\_PIN\_RS, LCD\_PIN\_E, LCD\_PIN\_D4, LCD\_PIN\_D5, LCD\_PIN\_D6, LCD\_PIN\_D7);

File wordsFile;

// ===== Scores =====

int score\_team1 = 0;

int score\_team2 = 0;

// ===== Display formatting =====

#define TOP\_TEXT\_LEN     14

#define BOTTOM\_TEXT\_LEN  16

String pad\_center(String text, uint8\_t width) {

  text.trim();

  if (text.length() > width) return "";

  uint8\_t leftPad  = (width - text.length()) / 2;

  uint8\_t rightPad = width - text.length() - leftPad;

  String s;

  for (uint8\_t i=0;i<leftPad;i++)  s += ' ';

  s += text;

  for (uint8\_t i=0;i<rightPad;i++) s += ' ';

  return s;

}

// ---- VALIDATOR (no padding)

bool canDisplayRaw(const String &raw) {

  String t = normalize\_ws(raw);

  if (t.length() == 0)       return false;     // blank

  if (t.startsWith("#"))     return false;     // comment

  if (t.indexOf(' ') < 0) {

    return (t.length() <= BOTTOM\_TEXT\_LEN);    // single token ≤16 ok

  }

  if (t.length() <= TOP\_TEXT\_LEN) return true; // whole phrase fits top

  int cut = t.lastIndexOf(' ', TOP\_TEXT\_LEN);

  if (cut < 0) return false;                   // first token too long

  String bottom = t.substring(cut + 1);

  bottom.trim();

  return (bottom.length() <= BOTTOM\_TEXT\_LEN); // bottom must fit

}

// ---- FORMATTER (padding + centering)

String format\_for\_lcd(String text) {

  text = normalize\_ws(text);

  auto pad = [](const String &t, uint8\_t width) {

    String s = t; s.trim();

    if (s.length() > width) return String("");

    uint8\_t L = (width - s.length()) / 2;

    uint8\_t R = width - s.length() - L;

    String out; for (uint8\_t i=0;i<L;i++) out+=' '; out+=s; for (uint8\_t i=0;i<R;i++) out+=' ';

    return out;

  };

  if (text.length() == 0) return pad("", TOP\_TEXT\_LEN) + pad("", BOTTOM\_TEXT\_LEN);

  if (text.indexOf(' ') < 0 && text.length() > TOP\_TEXT\_LEN && text.length() <= BOTTOM\_TEXT\_LEN) {

    // single token 15–16 -> bottom line centered

    return pad("", TOP\_TEXT\_LEN) + pad(text, BOTTOM\_TEXT\_LEN);

  }

  if (text.length() <= TOP\_TEXT\_LEN) {

    return pad(text, TOP\_TEXT\_LEN) + pad("", BOTTOM\_TEXT\_LEN);

  }

  int lastSpaceTop = text.lastIndexOf(' ', TOP\_TEXT\_LEN);

  if (lastSpaceTop < 0) return String("");

  String topPart = text.substring(0, lastSpaceTop);

  String botPart = text.substring(lastSpaceTop + 1);

  botPart.trim();

  if (botPart.length() > BOTTOM\_TEXT\_LEN) return String("");

  return pad(topPart, TOP\_TEXT\_LEN) + pad(botPart, BOTTOM\_TEXT\_LEN);

}

void lcdClearLine(byte row) {

  lcd.setCursor(0,row);

  for (byte i=0;i<16;i++) lcd.print(' ');

}

void showScoresAndText(const String &mainText) {

  String topWin = mainText.substring(0, TOP\_TEXT\_LEN);

  String bot    = mainText.substring(TOP\_TEXT\_LEN);

  lcd.setCursor(0,0);  lcd.print(score\_team1);

  lcd.setCursor(1,0);  lcd.print(topWin);

  lcd.setCursor(15,0); lcd.print(score\_team2);

  lcd.setCursor(0,1);  lcd.print(bot);

}

// ===== Debounced buttons =====

struct DebouncedButton {

  byte pin;

  byte lastAdvertised;

  byte curAdvertised;

  byte lastRead;

  unsigned long lastChange;

  void begin(byte p) { pin=p; pinMode(p, INPUT\_PULLUP); lastAdvertised=curAdvertised=lastRead=HIGH; lastChange=0; }

  void update() {

    byte s = digitalRead(pin);

    unsigned long now = millis();

    if (s != lastRead) lastChange = now;

    if (now - lastChange > 50) curAdvertised = s;

    lastRead = s;

  }

  bool justPressed()  { bool jp = (curAdvertised != lastAdvertised) && (curAdvertised == LOW);  lastAdvertised = curAdvertised; return jp; }

  bool justReleased() { bool jr = (curAdvertised != lastAdvertised) && (curAdvertised == HIGH); lastAdvertised = curAdvertised; return jr; }

  bool isPressed()    { return curAdvertised == LOW; }

};

DebouncedButton btnStart, btnT1, btnT2, btnNext, btnMute;

// ===== Game state =====

enum GAME\_STATE { READY, IN\_ROUND, GAME\_DONE };

GAME\_STATE gameState = READY;

bool muted = false;

String currentWord;

// ===== Beep timing (speeds up) =====

unsigned long beep\_frequency\_change\_interval\_millis = 15000;

unsigned long beep\_interval\_millis[] = {500, 500, 300, 200};

const int NUM\_BEEP\_INTERVALS = 4;

int cur\_beep\_interval = 0;

bool next\_is\_tic = true;

unsigned long last\_tictoc\_millis = 0;

unsigned long last\_beep\_speed\_change\_millis = 0;

// ===== Persistent deck (max 120) =====

const uint16\_t MAX\_WORDS = 120;

unsigned long wordOffsets[MAX\_WORDS]; // deck of offsets

uint16\_t wordCount = 0;               // size of current deck (<=120)

uint16\_t wordPos   = 0;               // next index to serve

uint32\_t displayableTotal = 0;        // total usable lines in file (computed when building deck)

bool deckBuilt = false;               // built yet?

void fisherYatesShuffleDeck() {

  if (wordCount <= 1) return;

  for (int i = wordCount - 1; i > 0; --i) {

    int j = random(i + 1);

    unsigned long tmp = wordOffsets[i];

    wordOffsets[i] = wordOffsets[j];

    wordOffsets[j] = tmp;

  }

}

// RNG seeding — use human timing jitter

void reseedRNG() {

  unsigned long t = micros() ^ (millis() << 16);

  randomSeed(t);

}

// Build a deck (once or when exhausted):

// - If total usable lines <=120: use ALL of them (wordCount = total), shuffled.

// - If >120: pick a random 120-sample via reservoir sampling.

// Returns false if no usable lines found.

bool buildDeckReservoir() {

  wordsFile.seek(0);

  displayableTotal = 0;

  wordCount = 0;

  while (true) {

    unsigned long startPos = wordsFile.position();

    String line = wordsFile.readStringUntil('\n');

    if (line.length() == 0 && !wordsFile.available()) break;

    String t = line; t.trim();

    if (t.length() == 0) continue;

    if (t.startsWith("#")) continue;

    if (!canDisplayRaw(t)) continue;

    displayableTotal++;

    if (wordCount < MAX\_WORDS) {

      wordOffsets[wordCount++] = startPos;

    } else {

      // reservoir sampling

      uint32\_t j = (uint32\_t)random(displayableTotal); // 0..displayableTotal-1

      if (j < MAX\_WORDS) wordOffsets[j] = startPos;

    }

  }

  if (displayableTotal == 0) return false;

  if (displayableTotal < MAX\_WORDS) wordCount = (uint16\_t)displayableTotal;

  fisherYatesShuffleDeck();

  wordPos = 0;

  deckBuilt = true;

  return true;

}

// function for displaying the word loading in the center. called in getNextWordFromDeck(). Shown both at the start of the game when the game is loading

// the first batch. Also shown if in the middle of a round, the current batch of 120 words is exhausted.

void showLoading() {

  // show "Loading..." centered-ish on the bottom row

  lcd.setCursor(0,1);

  lcd.print("                ");  // clear bottom

  lcd.setCursor(3,1);            // rough center for 10 chars

  lcd.print("Loading...");

}

// Serve next word; if deck exhausted, rebuild per rules

bool getNextWordFromDeck(String &out) {

  // Need a deck for the first time OR we’ve exhausted the current deck

  // if current 120 word deck is exhausted during a round, pause timer & show the loading splash for the brief pause

  if (!deckBuilt || wordPos >= wordCount) {

    unsigned long pauseStart = millis();

    showLoading();

    if (!deckBuilt) {

      // First-time build

      reseedRNG();

      if (!buildDeckReservoir()) return false;

    } else if (displayableTotal <= MAX\_WORDS) {

      // ≤120 usable total: reshuffle same full set after it’s all been shown

      fisherYatesShuffleDeck();

      wordPos = 0;

    } else {

      // >120 usable: build a fresh random 120-sample

      reseedRNG();

      if (!buildDeckReservoir()) return false;

    }

    // Don’t consume round time during the brief load

    unsigned long pauseDur = millis() - pauseStart;

    last\_tictoc\_millis += pauseDur;

    last\_beep\_speed\_change\_millis += pauseDur;

  }

  // Serve next word

  unsigned long off = wordOffsets[wordPos++];

  wordsFile.seek(off);

  String line = wordsFile.readStringUntil('\n');

  line.trim();

  out = line;

  return out.length() > 0;

}

// ===== Beeps =====

void beep\_tic()      {

  if (!muted)

  tone(SPEAKER\_PIN, 300, 30);

  }

void beep\_toc()      { if (!muted) tone(SPEAKER\_PIN, 300, 30); }

void beep\_times\_up() {

  if (!muted) {

    tone(SPEAKER\_PIN, 300, 300);

    delay(300);

    tone(SPEAKER\_PIN, 300, 300);

    delay(300);

    tone(SPEAKER\_PIN, 300, 300);

    delay(300);

    tone(SPEAKER\_PIN, 300, 300);

    delay(300);

    tone(SPEAKER\_PIN, 300, 300);

    delay(300);

    tone(SPEAKER\_PIN, 300, 300);

    }

  else { delay(900); }

}

void beep\_power\_on() { if (!muted) tone(SPEAKER\_PIN, 300, 30); }

void beep\_small()    { if (!muted) tone(SPEAKER\_PIN, 300, 30); }

void beep\_win\_game() {

  if (muted) return;

  for (int i = 0; i < 3; ++i) {

    tone(SPEAKER\_PIN, 300, 250);

    delay(100);

    tone(SPEAKER\_PIN, 400, 250);

    delay(100);

    tone(SPEAKER\_PIN, 500, 250);

    delay(100);

  }

}

// ===== Round flow =====

void showWord(const String &word) {

  String two = format\_for\_lcd(word);

  if (two.length() == 0) two = format\_for\_lcd("(too long)");

  showScoresAndText(two);

}

void startRound() {

  lcd.clear();

  // IMPORTANT: do NOT build/reshuffle deck here.

  // We keep the deck persistent across rounds and only rebuild when exhausted.

  gameState = IN\_ROUND;

  cur\_beep\_interval = 0;

  next\_is\_tic = true;

  last\_tictoc\_millis = 0;

  last\_beep\_speed\_change\_millis = millis();

  String w;

  if (!getNextWordFromDeck(w)) {

    lcd.clear(); lcd.setCursor(0,0); lcd.print(F("No words indexed"));

    lcd.setCursor(0,1); lcd.print(F("Check words.txt"));

    return;

  }

  currentWord = w;

  showWord(currentWord);

}

void endRound(bool timesUp=true) {

  if (timesUp) beep\_times\_up();

  gameState = READY;

  String two = format\_for\_lcd("Press Start");

  showScoresAndText(two);

}

void do\_tic\_toc() {

  unsigned long now = millis();

  if (now - last\_beep\_speed\_change\_millis > beep\_frequency\_change\_interval\_millis) {

    last\_beep\_speed\_change\_millis = now;

    if (++cur\_beep\_interval >= NUM\_BEEP\_INTERVALS) { endRound(true); return; }

  }

  if (now - last\_tictoc\_millis > beep\_interval\_millis[cur\_beep\_interval]) {

    if (next\_is\_tic) beep\_tic(); else beep\_toc();

    next\_is\_tic = !next\_is\_tic;

    last\_tictoc\_millis = now;

  }

}

// ===== Setup / Loop =====

void setup() {

  pinMode(TRANSISTOR\_POWER\_PIN, OUTPUT);

  digitalWrite(TRANSISTOR\_POWER\_PIN, HIGH);

  pinMode(LCD\_PIN\_BL, OUTPUT);

  digitalWrite(LCD\_PIN\_BL, HIGH);

  pinMode(SPEAKER\_PIN, OUTPUT);

  btnStart.begin(START\_STOP\_PIN);

  btnT1.begin(TEAM1\_PIN);

  btnT2.begin(TEAM2\_PIN);

  btnNext.begin(NEXT\_PIN);

  btnMute.begin(CATEGORY\_PIN);

  lcd.begin(16,2);

  lcdClearLine(0); lcdClearLine(1);

  lcd.setCursor(0,0); lcd.print(F("Loading SD..."));

  pinMode(SD\_PIN\_CS, OUTPUT);

  digitalWrite(SD\_PIN\_CS, HIGH);

  if (!SD.begin(SD\_PIN\_CS)) {

    lcd.setCursor(0,1); lcd.print(F("SD FAIL"));

    while (1) { }

  }

  wordsFile = SD.open("words.txt", FILE\_READ);

  if (!wordsFile) wordsFile = SD.open("WORDS.TXT", FILE\_READ);

  if (!wordsFile) {

    lcd.setCursor(0,1); lcd.print(F("words.txt?"));

    while (1) { }

  }

  // NOTE: We intentionally do NOT build the deck in setup().

  // This keeps the first batch truly random (we seed & build on first Start press).

  beep\_power\_on();

  String two = format\_for\_lcd("Press Start");

  showScoresAndText(two);

  gameState = READY;

  score\_team1 = 0; score\_team2 = 0;

}

void loop() {

  btnStart.update();

  btnT1.update();

  btnT2.update();

  btnNext.update();

  btnMute.update();

  if (btnMute.justPressed()) {

    muted = !muted;

    lcdClearLine(1);

    lcd.setCursor(4,1);

    lcd.print(muted ? F("Muted") : F("Sound On"));

    if (!muted) beep\_small();

    delay(300);

    if (gameState == IN\_ROUND) showWord(currentWord);

    else {

      String two = format\_for\_lcd("Press Start");

      showScoresAndText(two);

    }

  }

  switch (gameState) {

    case READY:

      if (btnStart.justPressed()) startRound();

      if (btnT1.justPressed()) {

        score\_team1++; beep\_small();

        if (score\_team1 == 7) {

          lcd.clear();

          lcd.setCursor(0,0);

          lcd.print(F("Team 1 Wins!"));

          beep\_win\_game();

          gameState = GAME\_DONE; }

        else {

          String two = format\_for\_lcd("Press Start");

          showScoresAndText(two);

          }

      }

      if (btnT2.justPressed()) {

        score\_team2++; beep\_small();

        if (score\_team2 == 7) {

          lcd.clear();

          lcd.setCursor(0,0);

          lcd.print(F("Team 2 Wins!"));

          beep\_win\_game();

          gameState = GAME\_DONE; }

        else { String two = format\_for\_lcd("Press Start");

        showScoresAndText(two);

        }

      }

      break;

    case IN\_ROUND:

      if (btnStart.justPressed()) { endRound(false); break; } // stop early

      if (btnNext.justPressed()) {

        String w;

        if (getNextWordFromDeck(w)) { currentWord = w; showWord(currentWord); }

      }

      do\_tic\_toc();

      break;

    case GAME\_DONE:

      if (btnStart.justPressed()) {

        score\_team1 = 0; score\_team2 = 0;

        String two = format\_for\_lcd("Press Start");

        showScoresAndText(two);

        gameState = READY;

      }

      break;

  }

  delay(5);

}

String normalize\_ws(String s) {

  // Convert tabs & NBSPs to regular spaces, drop stray UTF-8 lead byte (0xC2),

  // collapse multiple spaces, strip CRs.

  String out; out.reserve(s.length());

  bool inSpace = false;

  for (uint16\_t i = 0; i < s.length(); ++i) {

    unsigned char c = s[i];

    if (c == '\r') continue;            // remove CR

    if (c == 0xC2) continue;            // drop UTF-8 lead byte (e.g., for NBSP)

    if (c == '\t' || c == ' ' || c == 0xA0) { // tab, space, NBSP

      if (!inSpace) { out += ' '; inSpace = true; }

    } else {

      out += (char)c;

      inSpace = false;

    }

  }

  out.trim();

  return out;

}