/\*

  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

#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 windows =====

// Top row is 16 chars total but col0 & col15 are scores → 14-char window in the middle.

#define TOP\_TEXT\_LEN     14

#define BOTTOM\_TEXT\_LEN  16

// ============================================================================

// Utility: center-pad a string to given width. Returns "" if it won't fit.

// ============================================================================

String centerPad(const String &src, uint8\_t width) {

  String s = src;

  s.trim();

  if (s.length() > width) return String("");   // too long for this line

  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;

}

// ============================================================================

// One TRUE splitter used EVERYWHERE (indexing & rendering)

// - Implements the same behavior that worked in your old sketch.

// - Fills top/bot with UNPADDED text portions to print.

// - Returns true if the phrase can be displayed.

// ============================================================================

bool splitPhrase(const String &raw, String &top, String &bot) {

  String text = raw;

  text.trim();

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

  if (text[0] == '#')     return false;    // comment line

  int firstSpace = text.indexOf(' ');

  // Single-token cases

  if (firstSpace < 0) {

    if (text.length() <= TOP\_TEXT\_LEN)      { top = text; bot = "";  return true; }

    if (text.length() <= BOTTOM\_TEXT\_LEN)   { top = "";   bot = text; return true; }

    return false; // single token >=17 chars is not displayable

  }

  // Whole phrase fits on top window

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

    top = text; bot = "";

    return true;

  }

  // Split at the LAST space that keeps top within 14

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

  if (cut < 0) {

    // First token exceeds 14 and there's no space before 15th char → can't split nicely

    return false;

  }

  top = text.substring(0, cut);

  bot = text.substring(cut + 1);

  bot.trim();

  // Bottom must fit in 16

  if (bot.length() > BOTTOM\_TEXT\_LEN) return false;

  return true;

}

// ============================================================================

// Validator used by deck builder (delegates to splitPhrase so it's identical

// to what we actually show on screen).

// ============================================================================

bool canDisplayRaw(const String &raw) {

  String t, b;

  return splitPhrase(raw, t, b);

}

// ============================================================================

// LCD helpers

// ============================================================================

void lcdClearLine(byte row) {

  lcd.setCursor(0, row);

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

}

// Show centered text with scores in the top corners.

// - topTextCentered: exactly 14 characters (centered) for top window

// - botTextCentered: exactly 16 characters (centered) for bottom row

void showScoresAndTextCentered(const String &topTextCentered, const String &botTextCentered) {

  // Top scores & text window: [score\_team1][ 14 chars ][score\_team2]

  lcd.setCursor(0, 0);  lcd.print(score\_team1);            // left score at col0

  lcd.setCursor(1, 0);  lcd.print(topTextCentered);        // centered content in 14-char window

  lcd.setCursor(15, 0); lcd.print(score\_team2);            // right score at col15

  // Bottom: full-width centered text

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

}

// Render a word/phrase: split → center each line → draw

void showWord(const String &word) {

  String top, bot;

  if (!splitPhrase(word, top, bot)) {

    // Display a safe fallback if a line somehow slips through the index filter.

    top = "(too long)";

    bot = "";

  }

  // Center-pad each line to the exact window width.

  String topC = centerPad(top, TOP\_TEXT\_LEN);

  String botC = centerPad(bot, BOTTOM\_TEXT\_LEN);

  // As a last-resort safety (should not happen if splitPhrase returned true)

  if (topC.length() == 0) topC = centerPad("", TOP\_TEXT\_LEN);

  if (botC.length() == 0) botC = centerPad("", BOTTOM\_TEXT\_LEN);

  showScoresAndTextCentered(topC, botC);

}

// ============================================================================

// 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;     // 50 ms debounce

    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 over time)

// ============================================================================

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 (reservoir sampled, max 120 offsets stored)

// ============================================================================

const uint16\_t MAX\_WORDS = 120;

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

uint16\_t wordCount = 0;               // number of deck entries currently stored

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

uint32\_t displayableTotal = 0;        // count of usable lines in file (for sampling)

bool deckBuilt = false;               // whether we built at least once

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 (called on first Start or when deck is exhausted):

// - If total usable lines <= 120: use ALL of them; reshuffle when exhausted.

// - If > 120: reservoir-sample a random 120; when 'excludePrev' is true and

//             there are >=240 usable, avoid picking offsets from the previous deck.

bool buildDeckReservoir(bool excludePrev = false) {

  wordsFile.seek(0);

  displayableTotal = 0;   // total usable (split-able) lines in file

  wordCount        = 0;   // count we select into the deck (<=120)

  // First pass: sample from the "eligible" pool.

  // If excludePrev==true, eligible = usable && !inPrevDeck(startPos)

  // else eligible = usable.

  uint32\_t eligibleTotal = 0;

  while (true) {

    unsigned long startPos = wordsFile.position();

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

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

    String t = line; t.trim();

    if (t.length() == 0) continue;   // skip blanks

    if (t.startsWith("#")) continue; // skip comments

    // Only count "usable" lines (same rule as display)

    String tp, bp;

    if (!splitPhrase(t, tp, bp)) continue;

    // Track global usable total

    displayableTotal++;

    // Decide eligibility for this pass

    bool eligible = true;

    if (excludePrev && inPrevDeck(startPos)) eligible = false;

    if (eligible) {

      // reservoir over the eligible pool

      eligibleTotal++;

      if (wordCount < MAX\_WORDS) {

        wordOffsets[wordCount++] = startPos;

      } else {

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

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

      }

    }

  }

  // If we wanted to exclude the previous deck but couldn't fill 120,

  // do a second pass that allows everything to top up the remainder.

  if (excludePrev && wordCount < MAX\_WORDS) {

    wordsFile.seek(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;

      String tp, bp;

      if (!splitPhrase(t, tp, bp)) continue;

      // If we already selected this offset, skip; otherwise add until 120.

      bool already = false;

      for (uint16\_t i = 0; i < wordCount; ++i) {

        if (wordOffsets[i] == startPos) { already = true; break; }

      }

      if (already) continue;

      wordOffsets[wordCount++] = startPos;

      if (wordCount >= MAX\_WORDS) break;

    }

  }

  if (displayableTotal == 0) return false;

  // If there are <=120 usable lines in total, keep wordCount == displayableTotal

  if (displayableTotal < MAX\_WORDS) {

    wordCount = (uint16\_t)displayableTotal;

  }

  fisherYatesShuffleDeck();

  wordPos   = 0;

  deckBuilt = true;

  return true;

}

// Quick "Loading..." helper shown while (re)building deck

void showLoading() {

  lcd.setCursor(0, 1);

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

  lcd.setCursor(3, 1);            // rough center for "Loading..."

  lcd.print("Loading...");

}

// ===== Track the previous deck so we can avoid repeating it back-to-back =====

unsigned long prevDeckOffsets[MAX\_WORDS];

uint16\_t      prevDeckCount  = 0;

bool          prevDeckValid  = false;

inline bool inPrevDeck(unsigned long off) {

  if (!prevDeckValid) return false;

  for (uint16\_t i = 0; i < prevDeckCount; ++i) {

    if (prevDeckOffsets[i] == off) return true;

  }

  return false;

}

inline void snapshotPrevDeck() {

  // copy current deck into "previous deck"

  prevDeckCount = wordCount;

  for (uint16\_t i = 0; i < wordCount; ++i) prevDeckOffsets[i] = wordOffsets[i];

  prevDeckValid = (prevDeckCount > 0);

}

// Ensure a deck is present and ready to serve the NEXT word.

// - First build OR after we've consumed the current deck.

// - If total usable <=120: reshuffle same deck.

// - If >120: build a brand-new random 120-sample.

// Returns false on failure (no displayable lines).

bool ensureDeckReadyForServe() {

  // Need a deck for the first time OR we've consumed all entries

  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: 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;

    }

    // Pause compensation so round time doesn't shrink

    unsigned long pauseDur = millis() - pauseStart;

    last\_tictoc\_millis += pauseDur;

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

  }

  return true;

}

// Serve next word; (re)build deck if needed; skip any non-displayable line defensively.

// Serve next word; rebuild/reshuffle exactly at deck boundaries.

// - With >120 usable: after the 120-sample is consumed, we immediately build a new random 120.

// - With ≤120 usable: after all shown, we reshuffle and cycle again (no repeats within a cycle).

bool getNextWordFromDeck(String &out) {

  // Make sure a deck exists (first run) or, if we just exhausted, refresh per rules

  if (!ensureDeckReadyForServe()) return false;

  // Try to fetch a displayable word. We already filtered at build time,

  // but keep the check for safety and to skip any odd read.

  while (true) {

    // If we hit the end of the current deck, refresh per rules and continue.

    if (wordPos >= wordCount) {

      if (!ensureDeckReadyForServe()) return false;

    }

    unsigned long off = wordOffsets[wordPos++];

    wordsFile.seek(off);

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

    line.trim();

    if (line.length() == 0) continue;

    String tp, bp;

    if (splitPhrase(line, tp, bp)) {

      out = line;

      return true;

    }

    // If somehow not displayable, loop to try the next entry.

  }

}

// ============================================================================

// 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 startRound() {

  lcd.clear();

  // Do NOT build/reshuffle here manually; we build on demand inside getNextWordFromDeck()

  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)) {

    // If deck failed to provide a word, show message and return to READY

    lcd.clear();

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

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

    gameState = READY;             // ensure we don't run the timer/buzzer

    return;

  }

  currentWord = w;

  showWord(currentWord);

}

void endRound(bool timesUp = true) {

  if (timesUp) beep\_times\_up();

  gameState = READY;

  // Show "Press Start" centered

  String topC = centerPad("Press Start", TOP\_TEXT\_LEN);

  String botC = centerPad("", BOTTOM\_TEXT\_LEN);

  showScoresAndTextCentered(topC, botC);

}

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) { }

  }

  // We don't build the deck here. We build on first Start (inside getNextWordFromDeck()).

  beep\_power\_on();

  // Initial screen: "Press Start" centered with scores.

  String topC = centerPad("Press Start", TOP\_TEXT\_LEN);

  String botC = centerPad("", BOTTOM\_TEXT\_LEN);

  showScoresAndTextCentered(topC, botC);

  gameState = READY;

  score\_team1 = 0;

  score\_team2 = 0;

}

void loop() {

  btnStart.update();

  btnT1.update();

  btnT2.update();

  btnNext.update();

  btnMute.update();

  // Mute toggle feedback

  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 topC = centerPad("Press Start", TOP\_TEXT\_LEN);

      String botC = centerPad("", BOTTOM\_TEXT\_LEN);

      showScoresAndTextCentered(topC, botC);

    }

  }

  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 topC = centerPad("Press Start", TOP\_TEXT\_LEN);

          String botC = centerPad("", BOTTOM\_TEXT\_LEN);

          showScoresAndTextCentered(topC, botC);

        }

      }

      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 topC = centerPad("Press Start", TOP\_TEXT\_LEN);

          String botC = centerPad("", BOTTOM\_TEXT\_LEN);

          showScoresAndTextCentered(topC, botC);

        }

      }

      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 topC = centerPad("Press Start", TOP\_TEXT\_LEN);

        String botC = centerPad("", BOTTOM\_TEXT\_LEN);

        showScoresAndTextCentered(topC, botC);

        gameState = READY;

      }

      break;

  }

  delay(5);

}