```
<!doctype html>
<html>
<head>
<meta charset="utf-8"/>
<title>Playable Chess (Standalone)</title>
<style>
  body { font-family: system-ui, -apple-system, "Segoe UI", Roboto, Arial; display:flex; gap:20px;
        align-items:flex-start; padding:20px; background:#f3f3f3; }
  #boardCanvas { border: 4px solid #333; image-rendering: pixelated; cursor:pointer; }
  .panel { max-width:320px; }
  h2{margin:.2rem 0;}
  #moves { height:420px; overflow:auto; background:white; padding:10px; border-radius:8px;
border:1px solid #ddd; }
  button { margin-top:8px; padding:8px 10px; }
  .small { font-size:13px; color:#555; }
</style>
</head>
<body>

<canvas id="boardCanvas" width="640" height="640"></canvas>

<div class="panel">
  <h2>Chess — Playable</h2>
  <div class="small">Click a piece, then click a destination. Captures allowed. Promotion to
queen supported.</div>
  <p><strong>Turn:</strong> <span id="turnLabel">White</span></p>
  <div id="moves"></div>
  <button id="resetBtn">Reset</button>
  <p class="small">To embed in Canva: host this file and use Canva's Embed/Website element
to paste the page URL (Canva renders embeds in an iframe).</p>
</div>

<script>
/* Simple playable chessboard using Unicode piece glyphs.
   - Implements legal piece movements (no check detection).
   - Turn-based enforcement.
   - Pawn promotion (promote to queen).
   - Click to select / move.
   - Board stored as 8x8 array.
*/

// Unicode pieces
const glyph = {
  pw: "♙", rw: "♖", nw: "♘", bw: "♗", qw: "♕", kw: "♔",
  pb: "♟", rb: "♜", nb: "♞", bb: "♝", qb: "♛", kb: "♚"
};

const canvas = document.getElementById('boardCanvas');
const ctx = canvas.getContext('2d');
const size = canvas.width;
const sq = size/8;
const movesDiv = document.getElementById('moves');
const turnLabel = document.getElementById('turnLabel');
let selected = null;
let legalMovesCache = [];
let board = [];
let turn = 'w'; // 'w' or 'b'
const LIGHT = "#f0d9b5", DARK = "#b58863", HIGHLIGHT = "rgba(80,200,120,0.45)", ATTACK
= "rgba(200,80,80,0.45)";

function deepCopyBoard(b){
  return b.map(r => r.map(c => c ? {...c} : null));
}

function setupStartingBoard(){
  const empty = () => Array(8).fill(null);
  board = Array.from({length:8}, (_,r)=>{
    if(r===0) return [
      {t:'r',c:'b'},{t:'n',c:'b'},{t:'b',c:'b'},{t:'q',c:'b'},
      {t:'k',c:'b'},{t:'b',c:'b'},{t:'n',c:'b'},{t:'r',c:'b'}
    ];
    if(r===1) return Array(8).fill().map(()=>({t:'p',c:'b'}));
    if(r===6) return Array(8).fill().map(()=>({t:'p',c:'w'}));
    if(r===7) return [
      {t:'r',c:'w'},{t:'n',c:'w'},{t:'b',c:'w'},{t:'q',c:'w'},
      {t:'k',c:'w'},{t:'b',c:'w'},{t:'n',c:'w'},{t:'r',c:'w'}
    ];
    return empty();
  });
  turn = 'w';
  selected = null;
  legalMovesCache = [];
  updateUI();
}

function drawBoard(){
  ctx.clearRect(0,0,size,size);
  for(let r=0;r<8;r++){
    for(let f=0;f<8;f++){
      const isLight = (r+f)%2===0;
      ctx.fillStyle = isLight ? LIGHT : DARK;
      ctx.fillRect(f*sq, r*sq, sq, sq);
    }
  }
  // highlight legal moves
  if(selected){
    for(const m of legalMovesCache){
      ctx.fillStyle = m.capture ? ATTACK : HIGHLIGHT;
      ctx.fillRect(m.toF*sq, m.toR*sq, sq, sq);
    }
    // selected square ring
    ctx.strokeStyle = "#444";
    ctx.lineWidth = 3;
    ctx.strokeRect(selected.f*sq+2, selected.r*sq+2, sq-4, sq-4);
  }
  // draw pieces (unicode)
  ctx.textAlign = "center";
  ctx.textBaseline = "middle";
  ctx.font = (sq*0.72) + "px serif";
  for(let r=0;r<8;r++){
    for(let f=0;f<8;f++){
      const p = board[r][f];
      if(!p) continue;
      const key = p.t + p.c;
      const ch = glyph[key] || '?';
      ctx.fillStyle = p.c === 'w' ? "#111" : "#111";
      ctx.fillText(ch, f*sq + sq/2, r*sq + sq/2 + 4);
    }
  }
}

function inBounds(r,f){ return r>=0 && r<8 && f>=0 && f<8; }

function generateMovesFor(r,f){
  const p = board[r][f];
  if(!p) return [];
  const dir = p.c === 'w' ? -1 : 1;
  const moves = [];
  if(p.t === 'p'){
    // forward
    const fr = r + dir;
    if(inBounds(fr,f) && !board[fr][f]) {
      moves.push({fromR:r,fromF:f,toR:fr,toF:f,capture:false});
      // two squares on starting rank
      const startRank = p.c==='w' ? 6 : 1;
      if(r===startRank){
        const fr2 = r + dir*2;
        if(inBounds(fr2,f) && !board[fr2][f]) moves.push({fromR:r,fromF:f,toR:fr2,toF:f,capture:false,
double:true});
      }
    }
    // captures
    for(const df of [-1,1]){
      const tr = r + dir, tf = f + df;
      if(inBounds(tr,tf) && board[tr][tf] && board[tr][tf].c !== p.c){
        moves.push({fromR:r,fromF:f,toR:tr,toF:tf,capture:true});
      }
    }
    // en-passant NOT implemented (simple playable version)
  } else if(p.t === 'n'){
    const del = [[-2,-1],[-2,1],[-1,-2],[-1,2],[1,-2],[1,2],[2,-1],[2,1]];
    for(const [dr,df] of del){
      const tr=r+dr, tf=f+df;
      if(inBounds(tr,tf) && (!board[tr][tf] || board[tr][tf].c !== p.c)){
        moves.push({fromR:r,fromF:f,toR:tr,toF:tf,capture: !!board[tr][tf]});
      }
    }
  } else if(p.t === 'b' || p.t === 'r' || p.t === 'q'){
    const dirs = [];
    if(p.t === 'b' || p.t === 'q') dirs.push([-1,-1],[-1,1],[1,-1],[1,1]);
    if(p.t === 'r' || p.t === 'q') dirs.push([-1,0],[1,0],[0,-1],[0,1]);
    for(const [dr,df] of dirs){
      let tr=r+dr, tf=f+df;
      while(inBounds(tr,tf)){
        if(!board[tr][tf]){
          moves.push({fromR:r,fromF:f,toR:tr,toF:tf,capture:false});
        } else {
          if(board[tr][tf].c !== p.c) moves.push({fromR:r,fromF:f,toR:tr,toF:tf,capture:true});
          break;
        }
        tr+=dr; tf+=df;
      }
    }
  } else if(p.t === 'k'){
    for(let dr=-1;dr<=1;dr++) for(let df=-1;df<=1;df++){
      if(dr===0 && df===0) continue;
      const tr = r+dr, tf=f+df;
      if(inBounds(tr,tf) && (!board[tr][tf] || board[tr][tf].c !== p.c)){
        moves.push({fromR:r,fromF:f,toR:tr,toF:tf,capture: !!board[tr][tf]});
      }
    }
    // simple castling: NOT implemented to keep complexity down
  }
  return moves;
}

function allLegalMovesForColor(color){
  const list = [];
  for(let r=0;r<8;r++) for(let f=0;f<8;f++){
    const p = board[r][f];
    if(p && p.c === color){
      const ms = generateMovesFor(r,f);
      ms.forEach(m => list.push(m));
    }
  }
  return list;
}

function coordsFromEvent(e){
  const rect = canvas.getBoundingClientRect();
  const x = (e.clientX - rect.left) * (canvas.width / rect.width);
  const y = (e.clientY - rect.top) * (canvas.height / rect.height);
  const f = Math.floor(x / sq);
  const r = Math.floor(y / sq);
  return {r,f};
}

function updateUI(){
  drawBoard();
  turnLabel.textContent = turn === 'w' ? 'White' : 'Black';
  movesDiv.innerHTML = '';
}

canvas.addEventListener('click', (e)=>{
  const {r,f} = coordsFromEvent(e);
  if(!inBounds(r,f)) return;
  const p = board[r][f];

  // If selecting your own piece
  if(p && p.c === turn){
    selected = {r,f};
    legalMovesCache = generateMovesFor(r,f);
    updateUI();
    return;
  }

  // If there is a selected piece, try moving
  if(selected){
    const candidate = legalMovesCache.find(m => m.toR===r && m.toF===f);
    if(candidate){
      // execute move
      const from = selected;
      const piece = board[from.r][from.f];
      // move
      board[r][f] = piece;
      board[from.r][from.f] = null;
      // pawn promotion
      if(piece.t === 'p' && (r===0 || r===7)){
        // simple: auto promote to queen
        piece.t = 'q';
        alert((piece.c==='w'?'White':'Black') + " pawn promoted to Queen!");
      }
      // switch turn
      turn = (turn==='w') ? 'b' : 'w';
      selected = null;
      legalMovesCache = [];
      updateUI();
      logMove(piece, from, {r,f}, candidate.capture);
      return;
    } else {
      // click somewhere else: clear selection
      selected = null;
      legalMovesCache = [];
      updateUI();
    }
  }
});

function algebraic(from, to){
  const file = ['a','b','c','d','e','f','g','h'];
  return file[from.f] + (8-from.r) + '-' + file[to.f] + (8-to.r);
}

function logMove(piece, from, to, capture){
  const line = document.createElement('div');
  line.textContent = (piece.c==='w' ? 'W' : 'B') + ' ' + (piece.t.toUpperCase()) + ' ' +
algebraic(from,to) + (capture ? ' x' : '');
  movesDiv.prepend(line);
}

document.getElementById('resetBtn').addEventListener('click', setupStartingBoard);

// keyboard optional: press 'u' to undo last move? (not implemented)
setupStartingBoard();
</script>
</body>
</html>
```