```
    1: #!/bin/bash
    2: set -eo pipefail
    3:
    4: dir_script="$(cd "$(dirname "$0")" && pwd)"
    5: CFG="$dir_script/config.cfg"
    6:
    7: declare -a deck_idx
    8:
    9: _valida_entero_rango(){
   10:   local nombre="$1" val="$2" min="$3" max="$4"
   11:   val=${val%$'\r'}
   12:   case "$val" in
   13:     ''|*[!0-9]*) echo "$nombre debe ser entero"; return 1;;
   14:   esac
   15:   if [ "$val" -lt "$min" ] || [ "$val" -gt "$max" ]; then
   16:     echo "$nombre fuera de rango [$min..$max]"
   17:     return 1
   18:   fi
   19: }
   20:
   21: _valida_enum(){ local nombre="$1" val="$2" x
   22:   for x in $3; do [[ "$val" == "$x" ]] && return 0; done
   23:   echo "$nombre debe ser uno de: $3"; return 1
   24: }
   25: _valida_log_ruta(){
   26:   local ruta="$1" abs parent
   27:   [[ -n "$ruta" ]] || { echo "LOG vacio" >&2; return 1; }
   28:   [[ "$ruta" == */ ]] && { echo "LOG no puede terminar en /" >&2; return 1; }
   29:   [[ "$ruta" =~ [[:space:]] ]] && { echo "LOG no puede contener espacios" >&2; retur
n 1; }
   30:
   31:   if [[ "$ruta" = /* ]]; then abs="$ruta"; else abs="$dir_script/$ruta"; fi
   32:   parent="${abs%/*}"
   33:
   34:   [[ -d "$parent" ]] || { echo "Directorio de LOG inexistente: $parent" >&2; return
1; }
   35:   [[ -w "$parent" ]] || { echo "Sin permisos de escritura en: $parent" >&2; return 1
; }
   36:
   37:   if [[ -e "$abs" && ! -w "$abs" ]]; then
   38:     echo "Aviso: fichero LOG no escribible: $abs (se podra crear/rotar en el directo
rio)" >&2
   39:   fi
   40:   return 0
   41: }
   42:
   43: cargar_y_validar_cfg(){
   44:   if [ ! -f "$CFG" ]; then
   45:     echo "Falta $CFG" >&2; exit 1
   46:   fi
   47:   local GREP_E="/usr/xpg4/bin/grep"; [ -x "$GREP_E" ] || GREP_E="grep"
   48:   if ! "$GREP_E" -q '[^[:space:]]' "$CFG"; then
   49:     echo "$CFG esta vacio" >&2; exit 1
   50:   fi
   51:
   52:   local devuelto="JUGADORES PV ESTRATEGIA MAXIMO LOG"
   53:   local idx=1 line k v nlines=0
   54:   JUGADORES= PV= ESTRATEGIA= MAXIMO= LOG=
   55:   while IFS= read -r line || [ -n "$line" ]; do
   56:     line=${line%$'\r'}
   57:     case "$line" in *=*) ;; * ) echo "Linea invalida (sin '='): '$line'" >&2; exit 1
;; esac
   58:     k=${line%%=*}; v=${line#*=}
   59:     set -- $devuelto; eval "exp_k=\${$idx}"
   60:     if [ "$k" != "$exp_k" ]; then
   61:       echo "Orden/clave invalido en linea $idx: se esperaba '$exp_k', se leyo '$k'"
>&2; exit 1
   62:     fi
   63:     case "$v" in *[[:space:]]*) echo "Espacios no permitidos en el valor de $k" >&2;
```

```
 exit 1;;
   64:                     "" ) echo "Valor vacio en $k" >&2; exit 1;; esac
   65:     case "$k" in
   66:       JUGADORES)  JUGADORES="$v" ;;
   67:       PV)         PV="$v" ;;
   68:       ESTRATEGIA) ESTRATEGIA="$v" ;;
   69:       MAXIMO)     MAXIMO="$v" ;;
   70:       LOG)        LOG="$v" ;;
   71:     esac
   72:     nlines=$((nlines+1)); idx=$((idx+1))
   73:   done < "$CFG"
   74:
   75:   [ "$nlines" -eq 5 ] || { echo "$CFG debe tener exactamente 5 lineas (tiene $nlines
)" >&2; exit 1; }
   76:
   77:   _valida_entero_rango "JUGADORES" "$JUGADORES" 2 4 || exit 1
   78:   _valida_entero_rango "PV"        "$PV"        10 30 || exit 1
   79:   _valida_enum          "ESTRATEGIA" "$ESTRATEGIA" "0 1 2" || exit 1
   80:   _valida_entero_rango  "MAXIMO"   "$MAXIMO"    0 50 || exit 1
   81:   _valida_log_ruta      "$LOG" || exit 1
   82: }
   83:
   84: guardar_cfg(){
   85:   local tmp
   86:   tmp="$(mktemp /tmp/cfg.XXXXXX)" || { echo "No se pudo crear temporal" >&2; return
1; }
   87:   {
   88:     printf 'JUGADORES=%s\n'  "$JUGADORES"
   89:     printf 'PV=%s\n'         "$PV"
   90:     printf 'ESTRATEGIA=%s\n' "$ESTRATEGIA"
   91:     printf 'MAXIMO=%s\n'     "$MAXIMO"
   92:     printf 'LOG=%s\n'        "$LOG"
   93:   } > "$tmp" || { rm -f "$tmp"; echo "No se pudo escribir temporal" >&2; return 1; }
   94:   mv "$tmp" "$CFG" || { rm -f "$tmp"; echo "No se pudo reemplazar $CFG" >&2; return
1; }
   95: }
   96:
   97: configuracion(){
   98:   cargar_y_validar_cfg || return 1
   99:   echo "Configuracion actual:"
  100:   echo "  JUGADORES=$JUGADORES"
  101:   echo "  PV=$PV"
  102:   echo "  ESTRATEGIA=$ESTRATEGIA"
  103:   echo "  MAXIMO=$MAXIMO"
  104:   echo "  LOG=$LOG"
  105:   echo
  106:   while :; do
  107:     printf "Jugadores [2-4] (%s): " "$JUGADORES"; IFS= read -r nv
  108:     [ -n "$nv" ] && cand="$nv" || cand="$JUGADORES"
  109:     _valida_entero_rango "JUGADORES" "$cand" 2 4 && { JUGADORES="$cand"; break; }
  110:   done
  111:   while :; do
  112:     printf "Puntos de vida PV [10-30] (%s): " "$PV"; IFS= read -r nv
  113:     [ -n "$nv" ] && cand="$nv" || cand="$PV"
  114:     _valida_entero_rango "PV" "$cand" 10 30 && { PV="$cand"; break; }
  115:   done
  116:   while :; do
  117:     printf "Estrategia IA [0|1|2] (%s): " "$ESTRATEGIA"; IFS= read -r nv
  118:     [ -n "$nv" ] && cand="$nv" || cand="$ESTRATEGIA"
  119:     _valida_enum "ESTRATEGIA" "$cand" "0 1 2" && { ESTRATEGIA="$cand"; break; }
  120:   done
  121:   while :; do
  122:     printf "Puntos maximo para victoria [0-50] (0 desactiva) (%s): " "$MAXIMO"; IFS=
 read -r nv
  123:     [ -n "$nv" ] && cand="$nv" || cand="$MAXIMO"
  124:     _valida_entero_rango "MAXIMO" "$cand" 0 50 && { MAXIMO="$cand"; break; }
  125:   done
  126:   guardar_cfg || { echo "No se pudo guardar $CFG" >&2; return 1; }
  127:   echo "Configuracion actualizada en $CFG."
```

```
  128: }
  129:
  130: nombre_cartas(){
  131:   case "$1" in
  132:     ATK2)    echo "Espada corta (-2 PV)";;
  133:     ATK4)    echo "Espada larga (-4 PV)";;
  134:     ATK6)    echo "Hacha (-6 PV)";;
  135:     DEF4)    echo "Escudo basico (bloquea 4)";;
  136:     DEF6)    echo "Escudo reforzado (bloquea 6)";;
  137:     HEAL3)   echo "Curacion (+3 PV)";;
  138:     DRAW1)   echo "Robo de carta";;
  139:     COUNTER) echo "Contraataque";;
  140:     *)       echo "Carta desconocida? [$1]";;
  141:   esac
  142: }
  143:
  144: _hacer_array_mazo(){
  145:   local __name="$1"
  146:   eval "$__name=( 'ATK2' 'ATK4' 'ATK4' 'ATK6' 'DEF4' 'DEF4' 'DEF6' 'HEAL3' 'DRAW1' '
COUNTER' )"
  147: }
  148:
  149: _barajar_array(){
  150:   local __name="$1" n i j tmp
  151:   eval "n=\${#${__name}[@]}"
  152:   i=$((n-1))
  153:   while [ $i -gt 0 ]; do
  154:     j=$((RANDOM % (i+1)))
  155:     eval "tmp=\${${__name}[$i]}"
  156:     eval "${__name}[$i]=\${${__name}[$j]}"
  157:     eval "${__name}[$j]=\$tmp"
  158:     i=$((i-1))
  159:   done
  160: }
  161:
  162: init_decks(){
  163:   local p
  164:   for p in 0 1 2 3; do
  165:     if [ "$p" -lt "$JUGADORES" ]; then
  166:       _hacer_array_mazo "deck$p"
  167:       _barajar_array   "deck$p"
  168:       deck_idx[$p]=0
  169:     else
  170:       eval "deck$p=()"
  171:       deck_idx[$p]=0
  172:     fi
  173:   done
  174: }
  175:
  176: imprimir_mazo(){
  177:   local p="$1" i=0 len code
  178:   eval "len=\${#deck$p[@]}"
  179:   echo "Mazo jugador $((p+1)) ($len cartas):"
  180:   while [ "$i" -lt "$len" ]; do
  181:     eval "code=\${deck$p[$i]}"
  182:     printf "  [%02d] %s\n" "$i" "$(nombre_cartas "$code")"
  183:     i=$((i+1))
  184:   done
  185: }
  186:
  187: imprimir_mazo_restante(){
  188:   local p="$1" i code idx len
  189:   idx=${deck_idx[$p]:-0}
  190:   eval "len=\${#deck$p[@]}"
  191:   echo "Mazo jugador $((p+1)) restante ($(($len-idx)) cartas):"
  192:   for ((i=idx; i<len; i++)); do
  193:     eval "code=\${deck$p[$i]}"
  194:     printf "  [%02d] %s\n" "$((i-idx))" "$(nombre_cartas "$code")"
  195:   done
```

```
  196: }
  197:
  198: iniciar_jugadores(){
  199:   local i
  200:   for i in 0 1 2 3; do
  201:     if [ "$i" -lt "$JUGADORES" ]; then
  202:       pv[$i]="$PV"; escudo[$i]=0; contra[$i]=0; jugadas[$i]=0
  203:     else
  204:       pv[$i]=0; escudo[$i]=0; contra[$i]=0; jugadas[$i]=0
  205:     fi
  206:   done
  207: }
  208:
  209: anadir_carta_mano(){ local p=$1 c=$2; eval "hand$p+=(\"$c\")"; }
  210: eliminar_carta_usada(){ local p=$1 idx=$2; eval "unset hand$p[$idx]"; eval "hand$p=(
\"\${hand$p[@]}\")"; }
  211: alive(){ local p=$1; [ "${pv[$p]:-0}" -gt 0 ]; }
  212:
  213: reparto_mano_inicial(){
  214:   local p
  215:   for p in 0 1 2 3; do
  216:     if [ "$p" -lt "$JUGADORES" ]; then
  217:       eval "hand$p=(\"\${deck$p[@]:0:5}\")"
  218:       eval "deck$p=(\"\${deck$p[@]:5}\")"
  219:       deck_idx[$p]=0
  220:     else
  221:       eval "hand$p=()"
  222:       eval "deck$p=()"
  223:       deck_idx[$p]=0
  224:     fi
  225:   done
  226:   return 0
  227: }
  228:
  229: roba_carta(){
  230:   local p="$1" outvar="$2" idx len c
  231:   idx=${deck_idx[$p]:-0}
  232:   eval "len=\${#deck$p[@]}"
  233:   if [ "$idx" -ge "$len" ]; then
  234:     c=""
  235:   else
  236:     eval "c=\${deck$p[$idx]}"
  237:     deck_idx[$p]=$((idx+1))
  238:   fi
  239:   printf -v "$outvar" '%s' "$c"
  240: }
  241:
  242: imprimir_mano(){
  243:   local p="$1" i=0 len code
  244:   eval "len=\${#hand$p[@]}"
  245:   echo "Mano jugador $((p+1)) ($len cartas):"
  246:   while [ "$i" -lt "$len" ]; do
  247:     eval "code=\${hand$p[$i]}"
  248:     printf "  (%d) %s\n" "$((i+1))" "$(nombre_cartas "$code")"
  249:     i=$((i+1))
  250:   done
  251: }
  252:
  253: ia_escoger_carta(){
  254:   local p=$1 strat="$ESTRATEGIA" code
  255:   eval "local arr=(\"\${hand$p[@]}\")"
  256:   local choice=""
  257:   if [ "$strat" -eq 1 ]; then
  258:     for code in "${arr[@]}"; do [[ "$code" == ATK* ]] && { choice="$code"; break; };
 done
  259:   elif [ "$strat" -eq 2 ]; then
  260:     for code in "${arr[@]}"; do [[ "$code" == DEF* || "$code" == HEAL3 ]] && { choic
e="$code"; break; }; done
  261:   fi
```

```
  262:   [ -z "$choice" ] && choice="${arr[$((RANDOM % ${#arr[@]}))]}"
  263:   echo "$choice"
  264: }
  265:
  266: ia_escoger_jugador(){
  267:   local p=$1 i best=-1 bestpv=999999
  268:   for ((i=0;i<JUGADORES;i++)); do
  269:     [ "$i" -ne "$p" ] || continue
  270:     alive "$i" || continue
  271:     if [ "${pv[$i]}" -lt "$bestpv" ]; then bestpv="${pv[$i]}"; best="$i"; fi
  272:   done
  273:   echo "$best"
  274: }
  275:
  276: jugador_elegir_carta(){
  277:   local p=$1 len i c sel
  278:   eval "len=\${#hand$p[@]}"
  279:   while :; do
  280:     >&2 echo "Elige carta [1-$len]:"
  281:     for ((i=0;i<len;i++)); do eval "c=\${hand$p[$i]}"; >&2 echo "  $((i+1))) $(nombr
e_cartas "$c")"; done
  282:     IFS= read -r sel < /dev/tty
  283:     case "$sel" in ''|*[!0-9]*) continue;; esac
  284:     sel=$((sel-1))
  285:     [ "$sel" -ge 0 ] && [ "$sel" -lt "$len" ] && { echo "$sel"; return; }
  286:   done
  287: }
  288:
  289: jugador_elegir_jugador(){
  290:   local p=$1 in t
  291:   while :; do
  292:     >&2 echo -n "Elige objetivo (1..$JUGADORES, distinto de ti): "
  293:     IFS= read -r in < /dev/tty
  294:     case "$in" in ''|*[!0-9]*) continue;; esac
  295:     t=$((in-1))
  296:     if [ "$t" -ge 0 ] && [ "$t" -lt "$JUGADORES" ] && [ "$t" -ne "$p" ] && [ "${pv[$
t]:-0}" -gt 0 ]; then
  297:       echo "$t"; return
  298:     fi
  299:   done
  300: }
  301:
  302: aplicar_ataque(){ # src dst dmg
  303:   local src=$1 dst=$2 dmg=$3
  304:   local eff block c
  305:   eff=$dmg
  306:   block="${escudo[$dst]}"
  307:   c="${contra[$dst]}"
  308:   (( block += 0 ))
  309:   (( c += 0 ))
  310:
  311:   if (( block > 0 )); then
  312:     if (( block >= eff )); then
  313:       escudo[$dst]=$(( block - eff ))
  314:       eff=0
  315:     else
  316:       eff=$(( eff - block ))
  317:     fi
  318:     escudo[$dst]=0
  319:   fi
  320:
  321:   if (( eff > 0 )); then
  322:     if (( c > 0 )); then
  323:       local ret=$(( eff / 2 ))
  324:       pv[$src]=$(( pv[$src] - ret ))
  325:       contra[$dst]=0
  326:     fi
  327:    pv[$dst]=$(( pv[$dst] - eff ))
  328:
```

```
  329:   if (( pv[$dst] <= 0 )); then
  330:     escudo[$dst]=0
  331:     contra[$dst]=0
  332:     eval "hand$dst=()"
  333:   fi
  334:   fi
  335: }
  336:
  337: jugar_carta(){ # p code
  338:   local p=$1 code=$2 tgt
  339:   case "$code" in
  340:     ATK2)
  341:       tgt=$( [ "$p" -eq 0 ] && jugador_elegir_jugador "$p" || ia_escoger_jugador "$p
" )
  342:       [[ "$tgt" =~ ^[0-9]+$ ]] || { echo "No hay objetivos validos."; return; }
  343:       aplicar_ataque "$p" "$tgt" 2
  344:       echo "P$((p+1)) ataca a P$((tgt+1)) (-2)"
  345:       ;;
  346:     ATK4)
  347:       tgt=$( [ "$p" -eq 0 ] && jugador_elegir_jugador "$p" || ia_escoger_jugador "$p
" )
  348:       [[ "$tgt" =~ ^[0-9]+$ ]] || { echo "No hay objetivos validos."; return; }
  349:       aplicar_ataque "$p" "$tgt" 4
  350:       echo "P$((p+1)) ataca a P$((tgt+1)) (-4)"
  351:       ;;
  352:     ATK6)
  353:       tgt=$( [ "$p" -eq 0 ] && jugador_elegir_jugador "$p" || ia_escoger_jugador "$p
" )
  354:       [[ "$tgt" =~ ^[0-9]+$ ]] || { echo "No hay objetivos validos."; return; }
  355:       aplicar_ataque "$p" "$tgt" 6
  356:       echo "P$((p+1)) ataca a P$((tgt+1)) (-6)"
  357:       ;;
  358:     DEF4) escudo[$p]=$(( ${escudo[$p]:-0} + 4 )); echo "P$((p+1)) gana escudo +4";;
  359:     DEF6) escudo[$p]=$(( ${escudo[$p]:-0} + 6 )); echo "P$((p+1)) gana escudo +6";;
  360:     HEAL3) pv[$p]=$(( ${pv[$p]:-0} + 3 )); echo "P$((p+1)) se cura +3";;
  361:     DRAW1)
  362:       local nc; roba_carta "$p" nc
  363:       if [ -n "$nc" ]; then
  364:         anadir_carta_mano "$p" "$nc"
  365:         echo "P$((p+1)) roba 1 carta ($(nombre_cartas "$nc"))"
  366:       else
  367:         echo "P$((p+1)) intenta robar, pero no quedan cartas"
  368:       fi
  369:       ;;
  370:     COUNTER) contra[$p]=1; echo "P$((p+1)) prepara contraataque";;
  371:   esac
  372: }
  373:
  374: cartas_restantes(){
  375:   local s=0 p len
  376:   for ((p=0; p<JUGADORES; p++)); do
  377:     [ "${pv[$p]:-0}" -gt 0 ] || continue
  378:     eval "len=\${#deck$p[@]}"
  379:     s=$(( s + (len - ${deck_idx[$p]:-0}) ))
  380:   done
  381:   echo "$s"
  382: }
  383: cartas_mano(){
  384:   local s=0 p
  385:   for ((p=0; p<JUGADORES; p++)); do
  386:     [ "${pv[$p]:-0}" -gt 0 ] || continue
  387:     eval "s=\$(( s + \${#hand$p[@]} ))"
  388:   done
  389:   echo "$s"
  390: }
  391: revisar_fin(){
  392:   local alivec=0 last=-1 p
  393:   for ((p=0;p<JUGADORES;p++)); do
  394:     [ "${pv[$p]}" -gt 0 ] && { alivec=$((alivec+1)); last=$p; }
```

```
  395:   done
  396:   [ "$alivec" -eq 1 ] && { echo "WIN:$last"; return; }
  397:
  398:   if [ "$MAXIMO" -gt 0 ]; then
  399:     for ((p=0;p<JUGADORES;p++)); do
  400:       [ "${pv[$p]}" -ge "$MAXIMO" ] && { echo "WIN:$p"; return; }
  401:     done
  402:   fi
  403:
  404:   if [ "$(cartas_restantes)" -eq 0 ] && [ "$(cartas_mano)" -eq 0 ]; then
  405:     echo "DECKS_OUT"; return
  406:   fi
  407:
  408:   echo "CONT"
  409: }
  410:
  411:
  412: desempatar(){
  413:   local best=-1 bestpv=-999 bestjg=-999 p
  414:   for ((p=0;p<JUGADORES;p++)); do
  415:     if [ "${pv[$p]}" -gt "$bestpv" ] || { [ "${pv[$p]}" -eq "$bestpv" ] && [ "${juga
das[$p]:-0}" -gt "$bestjg" ]; }; then
  416:       bestpv=${pv[$p]}; bestjg=${jugadas[$p]:-0}; best=$p
  417:     fi
  418:   done
  419:   echo "$best"
  420: }
  421:
  422: estado_actual_jugaores(){
  423:   local i
  424:   echo "Estado:"
  425:   for ((i=0;i<JUGADORES;i++)); do
  426:     printf "  P%d: PV=%d ESC=%d\n" $((i+1)) "${pv[$i]}" "${escudo[$i]}"
  427:   done
  428: }
  429:
  430: finalizar_para_log(){
  431:   local winner=$1 tpo=$((SECONDS - start_ts)) fecha hora tcz tcm tcj p1=- p2=- p3=-
p4=-
  432:   fecha=$(date +%d%m%y); hora=$(date +%H:%M:%S)
  433:   tcz=$(cartas_restantes); tcm=$(cartas_mano)
  434:   local s=0 p; for ((p=0;p<JUGADORES;p++)); do s=$((s+${jugadas[$p]:-0})); done; tcj
=$s
  435:   [ "$JUGADORES" -ge 1 ] && p1=${pv[0]}
  436:   [ "$JUGADORES" -ge 2 ] && p2=${pv[1]}
  437:   [ "$JUGADORES" -ge 3 ] && p3=${pv[2]}
  438:   [ "$JUGADORES" -ge 4 ] && p4=${pv[3]}
  439:   printf "%s|%s|%s|%s|%s|%s|%s|%s|%s|%s|%s|%s|%s|%s|%s\n" \
  440:     "$fecha" "$hora" "$tpo" "$JUGADORES" "$PV" "$ESTRATEGIA" "$MAXIMO" "$((winner+1)
)" \
  441:     "$p1" "$p2" "$p3" "$p4" "$tcz" "$tcm" "$tcj" >> "$LOG"
  442:   echo "Ganador: Jugador $((winner+1))"
  443:   echo "PV finales: P1=$p1 P2=$p2 P3=$p3 P4=$p4"
  444:   echo "TCZ=$tcz TCM=$tcm TCJ=$tcj Tiempo=$tpo s (log: $LOG)"
  445: }
  446:
  447: bucle_turnos(){
  448:   start_ts=$SECONDS
  449:   local turn=1 pid sel code end hlen dcard
  450:   while :; do
  451:     echo; echo "--- Turno $turn ---"
  452:     for ((pid=0; pid<JUGADORES; pid++)); do
  453:       alive "$pid" || continue
  454:
  455:       eval "hlen=\${#hand$pid[@]}"
  456:       if [ "$hlen" -eq 0 ]; then
  457:         code=""
  458:         roba_carta "$pid" code
  459:         [ -n "$code" ] && anadir_carta_mano "$pid" "$code"
```

```
  460:         eval "hlen=\${#hand$pid[@]}"
  461:       fi
  462:         if [ "$hlen" -eq 0 ]; then
  463:         echo "P$((pid+1)) no tiene cartas."
  464:         end=$(revisar_fin)
  465:         if [ "$end" != "CONT" ]; then
  466:           if [[ "$end" == WIN:* ]]; then finalizar_para_log "${end#WIN:}"; return; f
i
  467:           if [ "$end" = "DECKS_OUT" ]; then local w; w=$(desempatar); finalizar_para
_log "$w"; return; fi
  468:         fi
  469:         continue
  470:       fi
  471:
  472:
  473:
  474:       if [ "$pid" -eq 0 ]; then
  475:         sel=$(jugador_elegir_carta "$pid")
  476:         eval "code=\${hand$pid[$sel]}"
  477:         eliminar_carta_usada "$pid" "$sel"
  478:       else
  479:         code=$(ia_escoger_carta "$pid")
  480:         eval '
  481:           for i in "${!hand'"$pid"'[@]}"; do
  482:             if [ "${hand'"$pid"'[$i]}" = "'"$code"'" ]; then unset hand'"$pid"'[$i];
 break; fi
  483:           done
  484:           hand'"$pid"'=("${hand'"$pid"'[@]}")
  485:         '
  486:       fi
  487:
  488:       echo "P$((pid+1)) juega $(nombre_cartas "$code")"
  489:       jugar_carta "$pid" "$code"
  490:       jugadas[$pid]=$(( ${jugadas[$pid]:-0}+1 ))
  491:       estado_actual_jugaores
  492:
  493:       dcard=""
  494:       roba_carta "$pid" dcard
  495:       if [ -n "$dcard" ]; then
  496:         anadir_carta_mano "$pid" "$dcard"
  497:         echo "P$((pid+1)) roba del mazo: $(nombre_cartas "$dcard")"
  498:       fi
  499:
  500:       end=$(revisar_fin)
  501:       if [ "$end" != "CONT" ]; then
  502:         if [[ "$end" == WIN:* ]]; then finalizar_para_log "${end#WIN:}"; return; fi
  503:         if [ "$end" = "DECKS_OUT" ]; then local w; w=$(desempatar); finalizar_para_l
og "$w"; return; fi
  504:       fi
  505:     done
  506:     turn=$((turn+1))
  507:   done
  508: }
  509:
  510: jugar(){
  511:   set +e
  512:   init_decks;
  513:   iniciar_jugadores;
  514:
  515:   echo "=== Mazos barajados (completos) ==="
  516:   for ((p=0; p<JUGADORES; p++)); do imprimir_mazo "$p"; done
  517:
  518:  reparto_mano_inicial;
  519:
  520:   echo; echo "=== Manos iniciales ==="
  521:   for ((p=0; p<JUGADORES; p++)); do imprimir_mano "$p"; done
  522:
  523:   echo; echo "=== Mazos restantes tras repartir ==="
  524:   for ((p=0; p<JUGADORES; p++)); do imprimir_mazo_restante "$p"; done
```

```
525:
526:   echo; echo "=== PV iniciales ==="
527:   for ((i=0;i<JUGADORES;i++)); do echo "Jugador $((i+1)): PV=${pv[$i]}"; done
528:
529:   bucle_turnos
530: }
531:
532: estadisticas(){
533:   local LOGFILE="$LOG"
534:
535:   if [ ! -f "$LOGFILE" ]; then
536:     echo "No existe el fichero de log: $LOGFILE"
537:     return 0
538:   fi
539:   if ! grep '[0-9]' "$LOGFILE"; then
540:     echo "No hay partidas registradas en $LOGFILE"
541:     return 0
542:   fi
543:
544:   local AWK="/usr/xpg4/bin/awk"; [ -x "$AWK" ] || AWK="awk"
545:
546:   "$AWK" -F'|' '
547:   function max(a,b){return a>b?a:b}
548:
549:   BEGIN{
550:     total=0; sum_tpo=0; min_tpo=1e9
551:     max_pv_overall=-1; max_tcj=-1; sum_tcj=0
552:     for(i=1;i<=4;i++) wins[i]=0
553:   }
554:
555:   $1 !~ /^[0-9]{6}$/ { next }
556:
557:   {
558:     total++
559:
560:     tpo = $3 + 0
561:     sum_tpo += tpo
562:
563:     ganador = $8 + 0
564:     if (ganador>=1 && ganador<=4) wins[ganador]++
565:
566:     if (tpo < min_tpo) { min_tpo=tpo; line_min=$0 }
567:
568:     maxpv_game=-1
569:     topcount=0
570:     pv[1]=$9; pv[2]=$10; pv[3]=$11; pv[4]=$12
571:     for(i=1;i<=4;i++){
572:       if(pv[i] != "-"){
573:         v = pv[i] + 0
574:         if(v > maxpv_game) maxpv_game=v
575:       }
576:     }
577:     for(i=1;i<=4;i++){
578:       if(pv[i] != "-"){
579:         v = pv[i] + 0
580:         if(v == maxpv_game) topcount++
581:       }
582:     }
583:     if (maxpv_game > max_pv_overall) { max_pv_overall=maxpv_game; line_maxpv=$0 }
584:
585:     if (topcount >= 2) { tie_count++; ties[tie_count]=$0 }
586:
587:     tcj = $15 + 0
588:     sum_tcj += tcj
589:     if (tcj > max_tcj) { max_tcj=tcj; line_maxtcj=$0 }
590:   }
591:
592:   END{
593:     if (total == 0) { print "No hay partidas validas en el log."; exit }
```

```
  594:
  595:     printf "================ ESTADISTICAS ================\n"
  596:     printf "Fichero: %s\n", "'"$LOGFILE"'"
  597:     printf "Partidas totales: %d\n", total
  598:     printf "Tiempo medio (s): %.2f\n", (sum_tpo/total)
  599:
  600:     split(line_min, a, "|")
  601:     printf "\n-- Partida mas corta --\n"
  602:     printf "Fecha %s Hora %s | TPO=%s | Jug=%s PV=%s Estr=%s PMax=%s | Ganador=%s |
P1=%s P2=%s P3=%s P4=%s | TCZ=%s TCM=%s TCJ=%s\n",
  603:            a[1],a[2],a[3],a[4],a[5],a[6],a[7],a[8],a[9],a[10],a[11],a[12],a[13],a[14
],a[15]
  604:
  605:     split(line_maxpv, b, "|")
  606:     printf "\n-- Partida con mayor PV final -- (PV max=%d)\n", max_pv_overall
  607:     printf "Fecha %s Hora %s | TPO=%s | Jug=%s PV=%s Estr=%s PMax=%s | Ganador=%s |
P1=%s P2=%s P3=%s P4=%s | TCZ=%s TCM=%s TCJ=%s\n",
  608:            b[1],b[2],b[3],b[4],b[5],b[6],b[7],b[8],b[9],b[10],b[11],b[12],b[13],b[14
],b[15]
  609:
  610:     printf "\n-- Victorias --\n"
  611:     for(i=1;i<=4;i++){
  612:       pct = (total>0)? (100.0*wins[i]/total) : 0
  613:       printf "Jugador %d: %d (%.1f%%)\n", i, wins[i], pct
  614:     }
  615:
  616:     printf "\n-- Partidas con empate por PV final --\n"
  617:     if (tie_count==0) {
  618:       printf "Ninguna.\n"
  619:     } else {
  620:       for(i=1;i<=tie_count;i++){
  621:         split(ties[i], t, "|")
  622:         printf "#%d -> Fecha %s Hora %s | Ganador=%s | PV finales: P1=%s P2=%s P3=%s
 P4=%s\n",
  623:                i, t[1], t[2], t[8], t[9], t[10], t[11], t[12]
  624:       }
  625:     }
  626:
  627:     printf "\n-- Extra --\n"
  628:     printf "TCJ medio: %.2f\n", (sum_tcj/total)
  629:     split(line_maxtcj, c, "|")
  630:     printf "Partida con mas jugadas (TCJ=%d): Fecha %s Hora %s | Ganador=%s\n",
  631:            max_tcj, c[1], c[2], c[8]
  632:   }' "$LOGFILE"
  633: }
  634: salir(){ echo "Saliendo del programa. Hasta pronto!"; exit 0; }
  635:
  636: menu(){
  637:   while true; do
  638:     echo "============================"
  639:     echo "      MENU PRINCIPAL       "
  640:     echo "============================"
  641:     echo "[C] Configuracion"
  642:     echo "[J] Jugar"
  643:     echo "[E] Estadisticas"
  644:     echo "[S] Salir"
  645:     echo "============================"
  646:     printf "Selecciona una opcion: "
  647:     IFS= read -r opcion
  648:     case "$opcion" in
  649:       [cC]) configuracion ;;
  650:       [jJ]) jugar ;;
  651:       [eE]) estadisticas ;;
  652:       [sS]) salir ;;
  653:       *)    echo "Opcion no valida. Intenta de nuevo." ;;
  654:     esac
  655:     echo
  656:     printf "Presiona Enter para continuar..."
  657:     IFS= read -r _
```

```
  done
}

cargar_y_validar_cfg || { echo "Config invalida. Corrige $CFG" >&2; exit 1; }
menu

```