```
-1.00  0 -1                                0:Source:dominion.c
-1.00  0:Graph:dominion.gcno
-1.00  0:Data:dominion.gcda
-1.00  0:Runs:1
-1.00  0:Programs:1
-1.00  1:#include "dominion.h"
-1.00  2:#include "dominion_helpers.h"
-1.00  3:#include "rngs.h"
-1.00  4:#include <stdio.h>
-1.00  5:#include <math.h>
-1.00  6:#include <stdlib.h>
-1.00  7:
-1.00  8:int compare(const void* a, const void* b) {
-1.00  9:  if (*(int*)a > *(int*)b)
-1.00  10:    return 1;
-1.00  11:  if (*(int*)a < *(int*)b)
-1.00  12:    return -1;
-1.00  13:  return 0;
-1.00  14:}
-1.00  15:
-1.00  16:struct gameState* newGame() {
-1.00  17:  struct gameState* g = malloc(sizeof(struct gameState));
-1.00  18:  return g;
-1.00  19:}
-1.00  20:
-1.00  21:int* kingdomCards(int k1, int k2, int k3, int k4, int k5, int k6, int k7,
-1.00  22:                                  int k8, int k9, int k10) {
-1.00  23:  int* k = malloc(10 * sizeof(int));
-1.00  24:  k[0] = k1;
-1.00  25:  k[1] = k2;
-1.00  26:  k[2] = k3;
-1.00  27:  k[3] = k4;
-1.00  28:  k[4] = k5;
-1.00  29:  k[5] = k6;
-1.00  30:  k[6] = k7;
-1.00  31:  k[7] = k8;
-1.00  32:  k[8] = k9;
-1.00  33:  k[9] = k10;
-1.00  34:  return k;
-1.00  35:}
-1.00  36:
-1.00  37:int initializeGame(int numPlayers, int kingdomCards[10], int randomSeed,
-1.00  38:                                  struct gameState *state) {
-1.00  39:
-1.00  40:  int i;
-1.00  41:  int j;
-1.00  42:  int it;
-1.00  43:  //set up random number generator
-1.00  44:  SelectStream(1);
-1.00  45:  PutSeed((long)randomSeed);
-1.00  46:
-1.00  47:  //check number of players
-1.00  48:  if (numPlayers > MAX_PLAYERS || numPlayers < 2)
-1.00  49:    {
-1.00  50:      return -1;
-1.00  51:    }
-1.00  52:
-1.00  53:  //set number of players
-1.00  54:  state->numPlayers = numPlayers;
-1.00  55:
-1.00  56:  //check selected kingdom cards are different
-1.00  57:  for (i = 0; i < 10; i++)
-1.00  58:    {
-1.00  59:      for (j = 0; j < 10; j++)
-1.00  60:        {
-1.00  61:                                if (j != i && kingdomCards[j] == kingdomCards[i])
-1.00  62:                                  {
-1.00  63:                                    return -1;
-1.00  64:                                  }
-1.00  65:        }
-1.00  66:    }
-1.00  67:
-1.00  68:
-1.00  69:  //initialize supply
-1.00  70:  ///////////////////////////////
-1.00  71:
-1.00  72:  //set number of Curse cards
-1.00  73:  if (numPlayers == 2)
-1.00  74:    {
-1.00  75:      state->supplyCount[curse] = 10;
-1.00  76:    }
```

```
-1.00  77:  else if (numPlayers == 3)
-1.00  78:    {
-1.00  79:      state->supplyCount[curse] = 20;
-1.00  80:    }
-1.00  81:  else
-1.00  82:    {
-1.00  83:      state->supplyCount[curse] = 30;
-1.00  84:    }
-1.00  85:
-1.00  86:  //set number of Victory cards
-1.00  87:  if (numPlayers == 2)
-1.00  88:    {
-1.00  89:      state->supplyCount[estate] = 8;
-1.00  90:      state->supplyCount[duchy] = 8;
-1.00  91:      state->supplyCount[province] = 8;
-1.00  92:    }
-1.00  93:  else
-1.00  94:    {
-1.00  95:      state->supplyCount[estate] = 12;
-1.00  96:      state->supplyCount[duchy] = 12;
-1.00  97:      state->supplyCount[province] = 12;
-1.00  98:    }
-1.00  99:
-1.00  100:  //set number of Treasure cards
-1.00  101:  state->supplyCount[copper] = 60 - (7 * numPlayers);
-1.00  102:  state->supplyCount[silver] = 40;
-1.00  103:  state->supplyCount[gold] = 30;
-1.00  104:
-1.00  105:  //set number of Kingdom cards
-1.00  106:  for (i = adventurer; i <= treasure_map; i++)                    //loop all cards
-1.00  107:    {
-1.00  108:      for (j = 0; j < 10; j++)                                                                                    //loop chosen cards
-1.00  109:                                                                {
-1.00  110:                                                                  if (kingdomCards[j] == i)
-1.00  111:                                                                    {
-1.00  112:                                                                      //check if card is a 'Victory' Kingdom card
-1.00  113:                                                                      if (kingdomCards[j] == great_hall || kingdomCards[j] == gardens)
-1.00  114:                                                                                                                  {
-1.00  115:                                                                                                                    if (numPlayers == 2){
-1.00  116:                                                                                                                      state->supplyCount[i] = 8;
-1.00  117:                                                                                                                    }
-1.00  118:                                                                                                                    else{ state->supplyCount[i] = 12; }
-1.00  119:                                                                                                                  }
-1.00  120:                                                                      else
-1.00  121:                                                                                                                  {
-1.00  122:                                                                                                                    state->supplyCount[i] = 10;
-1.00  123:                                                                                                                  }
-1.00  124:                                                                      break;
-1.00  125:                                                                    }
-1.00  126:                                                                  else    //card is not in the set choosen for the game
-1.00  127:                                                                    {
-1.00  128:                                                                      state->supplyCount[i] = -1;
-1.00  129:                                                                    }
-1.00  130:                                                                }
-1.00  131:
-1.00  132:    }
-1.00  133:
-1.00  134:  //////////////////////////
-1.00  135:  //supply intilization complete
-1.00  136:
-1.00  137:  //set player decks
-1.00  138:  for (i = 0; i < numPlayers; i++)
-1.00  139:    {
-1.00  140:      state->deckCount[i] = 0;
-1.00  141:      for (j = 0; j < 3; j++)
-1.00  142:                                                                {
-1.00  143:                                                                  state->deck[i][j] = estate;
-1.00  144:                                                                  state->deckCount[i]++;
-1.00  145:                                                                }
-1.00  146:      for (j = 3; j < 10; j++)
-1.00  147:                                                                {
-1.00  148:                                                                  state->deck[i][j] = copper;
-1.00  149:                                                                  state->deckCount[i]++;
-1.00  150:                                                                }
-1.00  151:    }
-1.00  152:
-1.00  153:  //shuffle player decks
-1.00  154:  for (i = 0; i < numPlayers; i++)
-1.00  155:    {
-1.00  156:      if ( shuffle(i, state) < 0 )
-1.00  157:      {
-1.00  158:                                                                  return -1;
```

```
-1.00 159:     }
-1.00 160:   }
-1.00 161:
-1.00 162:  //draw player hands
-1.00 163:  for (i = 0; i < numPlayers; i++)
-1.00 164:   {
-1.00 165:     //initialize hand size to zero
-1.00 166:     state->handCount[i] = 0;
-1.00 167:     state->discardCount[i] = 0;
-1.00 168:     //draw 5 cards
-1.00 169:     // for (j = 0; j < 5; j++)
-1.00 170:     //  {
-1.00 171:     //    drawCard(i, state);
-1.00 172:     //  }
-1.00 173:   }
-1.00 174:
-1.00 175:  //set embargo tokens to 0 for all supply piles
-1.00 176:  for (i = 0; i <= treasure_map; i++)
-1.00 177:   {
-1.00 178:     state->embargoTokens[i] = 0;
-1.00 179:   }
-1.00 180:
-1.00 181:  //initialize first player's turn
-1.00 182:  state->outpostPlayed = 0;
-1.00 183:  state->phase = 0;
-1.00 184:  state->numActions = 1;
-1.00 185:  state->numBuys = 1;
-1.00 186:  state->playedCardCount = 0;
-1.00 187:  state->whoseTurn = 0;
-1.00 188:  state->handCount[state->whoseTurn] = 0;
-1.00 189:  //int it; move to top
-1.00 190:
-1.00 191:  //Moved draw cards to here, only drawing at the start of a turn
-1.00 192:  for (it = 0; it < 5; it++){
-1.00 193:    drawCard(state->whoseTurn, state);
-1.00 194:  }
-1.00 195:
-1.00 196:  updateCoins(state->whoseTurn, state, 0);
-1.00 197:
-1.00 198:  return 0;
-1.00 199:}
-1.00 200:
-1.00 201:int shuffle(int player, struct gameState *state) {
-1.00 202:
-1.00 203:
-1.00 204:  int newDeck[MAX_DECK];
-1.00 205:  int newDeckPos = 0;
-1.00 206:  int card;
-1.00 207:  int i;
-1.00 208:
-1.00 209:  if (state->deckCount[player] < 1)
-1.00 210:    return -1;
-1.00 211:  qsort ((void*)(state->deck[player]), state->deckCount[player], sizeof(int), compare);
-1.00 212:  /* SORT CARDS IN DECK TO ENSURE DETERMINISM! */
-1.00 213:
-1.00 214:  while (state->deckCount[player] > 0) {
-1.00 215:    card = floor(Random() * state->deckCount[player]);
-1.00 216:    newDeck[newDeckPos] = state->deck[player][card];
-1.00 217:    newDeckPos++;
-1.00 218:    for (i = card; i < state->deckCount[player]-1; i++) {
-1.00 219:      state->deck[player][i] = state->deck[player][i+1];
-1.00 220:    }
-1.00 221:    state->deckCount[player]--;
-1.00 222:  }
-1.00 223:  for (i = 0; i < newDeckPos; i++) {
-1.00 224:    state->deck[player][i] = newDeck[i];
-1.00 225:    state->deckCount[player]++;
-1.00 226:  }
-1.00 227:
-1.00 228:  return 0;
-1.00 229:}
-1.00 230:
 0.50 231:int playCard(int handPos, int choice1, int choice2, int choice3, struct gameState *state)
-1.00 232:{
-1.00 233:  int card;
 0.50 234:  int coin_bonus = 0;                                                    //tracks coins gain from actions
-1.00 235:
-1.00 236:  //check if it is the right phase
 0.50 237:  if (state->phase != 0)
-1.00 238:   {
 0.00 239:     return -1;
-1.00 240:   }
```

```
-1.00 241:
-1.00 242:  //check if player has enough actions
 0.84 243:  if ( state->numActions < 1 )
-1.00 244:    {
 0.00 245:      return -1;
-1.00 246:    }
-1.00 247:
-1.00 248:  //get card played
 0.87 249:  card = handCard(handPos, state);
-1.00 250:
-1.00 251:  //check if selected card is an action
 0.87 252:  if ( card < adventurer || card > treasure_map )
-1.00 253:    {
-1.00 254:      return -1;
-1.00 255:    }
-1.00 256:
-1.00 257:  //play card
 0.87 258:  if ( cardEffect(card, choice1, choice2, choice3, state, handPos, &coin_bonus) < 0 )
-1.00 259:    {
 0.00 260:      return -1;
-1.00 261:    }
-1.00 262:
-1.00 263:  //reduce number of actions
 0.96 264:  state->numActions--;
-1.00 265:
-1.00 266:  //update coins (Treasure cards may be added with card draws)
 0.96 267:  updateCoins(state->whoseTurn, state, coin_bonus);
-1.00 268:
 0.96 269:  return 0;
-1.00 270:}
-1.00 271:
-1.00 272:int buyCard(int supplyPos, struct gameState *state) {
-1.00 273:  int who;
-1.00 274:  if (DEBUG){
-1.00 275:    printf("Entering buyCard...\n");
-1.00 276:  }
-1.00 277:
-1.00 278:  // I don't know what to do about the phase thing.
-1.00 279:
-1.00 280:  who = state->whoseTurn;
-1.00 281:
-1.00 282:  if (state->numBuys < 1){
-1.00 283:    if (DEBUG)
-1.00 284:      printf("You do not have any buys left\n");
-1.00 285:    return -1;
-1.00 286:  } else if (supplyCount(supplyPos, state) <1){
-1.00 287:    if (DEBUG)
-1.00 288:      printf("There are not any of that type of card left\n");
-1.00 289:    return -1;
-1.00 290:  } else if (state->coins < getCost(supplyPos)){
-1.00 291:    if (DEBUG)
-1.00 292:      printf("You do not have enough money to buy that. You have %d coins.\n", state->coins);
-1.00 293:    return -1;
-1.00 294:  } else {
-1.00 295:    state->phase=1;
-1.00 296:    //state->supplyCount[supplyPos]--;
-1.00 297:    gainCard(supplyPos, state, 0, who); //card goes in discard, this might be wrong.. (2 means goes into hand, 0 goes into discard)
-1.00 298:
-1.00 299:    state->coins = (state->coins) - (getCost(supplyPos));
-1.00 300:    state->numBuys--;
-1.00 301:    if (DEBUG)
-1.00 302:      printf("You bought card number %d for %d coins. You now have %d buys and %d coins.\n", supplyPos, getCost(supplyPos), state->numBuys, state->coins);
-1.00 303:  }
-1.00 304:
-1.00 305:  //state->discard[who][state->discardCount[who]] = supplyPos;
-1.00 306:  //state->discardCount[who]++;
-1.00 307:
-1.00 308:  return 0;
-1.00 309:}
-1.00 310:
-1.00 311:int numHandCards(struct gameState *state) {
-1.00 312:  return state->handCount[ whoseTurn(state) ];
-1.00 313:}
-1.00 314:
 0.87 315:int handCard(int handPos, struct gameState *state) {
 0.87 316:  int currentPlayer = whoseTurn(state);
 0.87 317:  return state->hand[currentPlayer][handPos];
-1.00 318:}
-1.00 319:
 0.96 320:int supplyCount(int card, struct gameState *state) {
 0.96 321:  return state->supplyCount[card];
-1.00 322:}
```

```
-1.00  323:
-1.00  324:int fullDeckCount(int player, int card, struct gameState *state) {
-1.00  325:  int i;
-1.00  326:  int count = 0;
-1.00  327:
-1.00  328:  for (i = 0; i < state->deckCount[player]; i++)
-1.00  329:    {
-1.00  330:      if (state->deck[player][i] == card) count++;
-1.00  331:    }
-1.00  332:
-1.00  333:  for (i = 0; i < state->handCount[player]; i++)
-1.00  334:    {
-1.00  335:      if (state->hand[player][i] == card) count++;
-1.00  336:    }
-1.00  337:
-1.00  338:  for (i = 0; i < state->discardCount[player]; i++)
-1.00  339:    {
-1.00  340:      if (state->discard[player][i] == card) count++;
-1.00  341:    }
-1.00  342:
-1.00  343:  return count;
-1.00  344:}
-1.00  345:
 0.87  346:int whoseTurn(struct gameState *state) {
 0.87  347:  return state->whoseTurn;
-1.00  348:}
-1.00  349:
-1.00  350:int endTurn(struct gameState *state) {
-1.00  351:  int k;
-1.00  352:  int i;
-1.00  353:  int currentPlayer = whoseTurn(state);
-1.00  354:
-1.00  355:  //Discard hand
-1.00  356:  for (i = 0; i < state->handCount[currentPlayer]; i++){
-1.00  357:    state->discard[currentPlayer][state->discardCount[currentPlayer]++] = state->hand[currentPlayer][i];//Discard
-1.00  358:    state->hand[currentPlayer][i] = -1;//Set card to -1
-1.00  359:  }
-1.00  360:  state->handCount[currentPlayer] = 0;//Reset hand count
-1.00  361:
-1.00  362:  //Code for determining the player
-1.00  363:  if (currentPlayer < (state->numPlayers - 1)){
-1.00  364:    state->whoseTurn = currentPlayer + 1;//Still safe to increment
-1.00  365:  }
-1.00  366:  else{
-1.00  367:    state->whoseTurn = 0;//Max player has been reached, loop back around to player 1
-1.00  368:  }
-1.00  369:
-1.00  370:  state->outpostPlayed = 0;
-1.00  371:  state->phase = 0;
-1.00  372:  state->numActions = 1;
-1.00  373:  state->coins = 0;
-1.00  374:  state->numBuys = 1;
-1.00  375:  state->playedCardCount = 0;
-1.00  376:  state->handCount[state->whoseTurn] = 0;
-1.00  377:
-1.00  378:  //int k; move to top
-1.00  379:  //Next player draws hand
-1.00  380:  for (k = 0; k < 5; k++){
-1.00  381:    drawCard(state->whoseTurn, state);//Draw a card
-1.00  382:  }
-1.00  383:
-1.00  384:  //Update money
-1.00  385:  updateCoins(state->whoseTurn, state , 0);
-1.00  386:
-1.00  387:  return 0;
-1.00  388:}
-1.00  389:
-1.00  390:int isGameOver(struct gameState *state) {
-1.00  391:  int i;
-1.00  392:  int j;
-1.00  393:
-1.00  394:  //if stack of Province cards is empty, the game ends
-1.00  395:  if (state->supplyCount[province] == 0)
-1.00  396:    {
-1.00  397:      return 1;
-1.00  398:    }
-1.00  399:
-1.00  400:  //if three supply pile are at 0, the game ends
-1.00  401:  j = 0;
-1.00  402:  for (i = 0; i < 25; i++)
-1.00  403:    {
-1.00  404:      if (state->supplyCount[i] == 0)
```

```
-1.00  405:        {
-1.00  406:          j++;
-1.00  407:        }
-1.00  408:    }
-1.00  409:  if ( j >= 3)
-1.00  410:    {
-1.00  411:      return 1;
-1.00  412:    }
-1.00  413:
-1.00  414:  return 0;
-1.00  415:}
-1.00  416:
-1.00  417:int scoreFor (int player, struct gameState *state) {
-1.00  418:
-1.00  419:  int i;
-1.00  420:  int score = 0;
-1.00  421:  //score from hand
-1.00  422:  for (i = 0; i < state->handCount[player]; i++)
-1.00  423:    {
-1.00  424:      if (state->hand[player][i] == curse) { score = score - 1; };
-1.00  425:      if (state->hand[player][i] == estate) { score = score + 1; };
-1.00  426:      if (state->hand[player][i] == duchy) { score = score + 3; };
-1.00  427:      if (state->hand[player][i] == province) { score = score + 6; };
-1.00  428:      if (state->hand[player][i] == great_hall) { score = score + 1; };
-1.00  429:      if (state->hand[player][i] == gardens) { score = score + ( fullDeckCount(player, 0, state) / 10 ); };
-1.00  430:    }
-1.00  431:
-1.00  432:  //score from discard
-1.00  433:  for (i = 0; i < state->discardCount[player]; i++)
-1.00  434:    {
-1.00  435:      if (state->discard[player][i] == curse) { score = score - 1; };
-1.00  436:      if (state->discard[player][i] == estate) { score = score + 1; };
-1.00  437:      if (state->discard[player][i] == duchy) { score = score + 3; };
-1.00  438:      if (state->discard[player][i] == province) { score = score + 6; };
-1.00  439:      if (state->discard[player][i] == great_hall) { score = score + 1; };
-1.00  440:      if (state->discard[player][i] == gardens) { score = score + ( fullDeckCount(player, 0, state) / 10 ); };
-1.00  441:    }
-1.00  442:
-1.00  443:  //score from deck
-1.00  444:  for (i = 0; i < state->discardCount[player]; i++)
-1.00  445:    {
-1.00  446:      if (state->deck[player][i] == curse) { score = score - 1; };
-1.00  447:      if (state->deck[player][i] == estate) { score = score + 1; };
-1.00  448:      if (state->deck[player][i] == duchy) { score = score + 3; };
-1.00  449:      if (state->deck[player][i] == province) { score = score + 6; };
-1.00  450:      if (state->deck[player][i] == great_hall) { score = score + 1; };
-1.00  451:      if (state->deck[player][i] == gardens) { score = score + ( fullDeckCount(player, 0, state) / 10 ); };
-1.00  452:    }
-1.00  453:
-1.00  454:  return score;
-1.00  455:}
-1.00  456:
-1.00  457:int getWinners(int players[MAX_PLAYERS], struct gameState *state) {
-1.00  458:  int i;
-1.00  459:  int j;
-1.00  460:  int highScore;
-1.00  461:  int currentPlayer;
-1.00  462:
-1.00  463:  //get score for each player
-1.00  464:  for (i = 0; i < MAX_PLAYERS; i++)
-1.00  465:    {
-1.00  466:      //set unused player scores to -9999
-1.00  467:      if (i >= state->numPlayers)
-1.00  468:        {
-1.00  469:          players[i] = -9999;
-1.00  470:        }
-1.00  471:      else
-1.00  472:        {
-1.00  473:          players[i] = scoreFor (i, state);
-1.00  474:        }
-1.00  475:    }
-1.00  476:
-1.00  477:  //find highest score
-1.00  478:  j = 0;
-1.00  479:  for (i = 0; i < MAX_PLAYERS; i++)
-1.00  480:    {
-1.00  481:      if (players[i] > players[j])
-1.00  482:        {
-1.00  483:          j = i;
-1.00  484:        }
-1.00  485:    }
-1.00  486:  highScore = players[j];
```

```
-1.00 487:
-1.00 488: //add 1 to players who had less turns
-1.00 489: currentPlayer = whoseTurn(state);
-1.00 490: for (i = 0; i < MAX_PLAYERS; i++)
-1.00 491:   {
-1.00 492:     if ( players[i] == highScore && i > currentPlayer )
-1.00 493:                                                                 {
-1.00 494:                                                                   players[i]++;
-1.00 495:                                                                 }
-1.00 496:   }
-1.00 497:
-1.00 498: //find new highest score
-1.00 499: j = 0;
-1.00 500: for (i = 0; i < MAX_PLAYERS; i++)
-1.00 501:   {
-1.00 502:     if ( players[i] > players[j] )
-1.00 503:                                                                 {
-1.00 504:                                                                   j = i;
-1.00 505:                                                                 }
-1.00 506:   }
-1.00 507: highScore = players[j];
-1.00 508:
-1.00 509: //set winners in array to 1 and rest to 0
-1.00 510: for (i = 0; i < MAX_PLAYERS; i++)
-1.00 511:   {
-1.00 512:     if ( players[i] == highScore )
-1.00 513:                                                                 {
-1.00 514:                                                                   players[i] = 1;
-1.00 515:                                                                 }
-1.00 516:     else
-1.00 517:                                                                 {
-1.00 518:                                                                   players[i] = 0;
-1.00 519:                                                                 }
-1.00 520:   }
-1.00 521:
-1.00 522: return 0;
-1.00 523:}
-1.00 524:
-1.00 525:int drawCard(int player, struct gameState *state)
-1.00 526:{                                                                 int count;
-1.00 527: int deckCounter;
-1.00 528: if (state->deckCount[player] <= 0){//Deck is empty
-1.00 529:
-1.00 530:   //Step 1 Shuffle the discard pile back into a deck
-1.00 531:   int i;
-1.00 532:   //Move discard to deck
-1.00 533:   for (i = 0; i < state->discardCount[player];i++){
-1.00 534:     state->deck[player][i] = state->discard[player][i];
-1.00 535:     state->discard[player][i] = -1;
-1.00 536:   }
-1.00 537:
-1.00 538:   state->deckCount[player] = state->discardCount[player];
-1.00 539:   state->discardCount[player] = 0;//Reset discard
-1.00 540:
-1.00 541:   //Shufffle the deck
-1.00 542:   shuffle(player, state);//Shuffle the deck up and make it so that we can draw
-1.00 543:
-1.00 544:   if (DEBUG){//Debug statements
-1.00 545:     printf("Deck count now: %d\n", state->deckCount[player]);
-1.00 546:   }
-1.00 547:
-1.00 548:   state->discardCount[player] = 0;
-1.00 549:
-1.00 550:   //Step 2 Draw Card
-1.00 551:   count = state->handCount[player];//Get current player's hand count
-1.00 552:
-1.00 553:   if (DEBUG){//Debug statements
-1.00 554:     printf("Current hand count: %d\n", count);
-1.00 555:   }
-1.00 556:
-1.00 557:   deckCounter = state->deckCount[player];//Create a holder for the deck count
-1.00 558:
-1.00 559:   if (deckCounter == 0)
-1.00 560:     return -1;
-1.00 561:
-1.00 562:   state->hand[player][count] = state->deck[player][deckCounter - 1];//Add card to hand
-1.00 563:   state->deckCount[player]--;
-1.00 564:   state->handCount[player]++;//Increment hand count
-1.00 565: }
-1.00 566:
-1.00 567: else{
-1.00 568:   int count = state->handCount[player];//Get current hand count for player
```

```
-1.00 569:   int deckCounter;
-1.00 570:   if (DEBUG){//Debug statements
-1.00 571:     printf("Current hand count: %d\n", count);
-1.00 572:   }
-1.00 573:
-1.00 574:   deckCounter = state->deckCount[player];//Create holder for the deck count
-1.00 575:   state->hand[player][count] = state->deck[player][deckCounter - 1];//Add card to the hand
-1.00 576:   state->deckCount[player]--;
-1.00 577:   state->handCount[player]++;//Increment hand count
-1.00 578: }
-1.00 579:
-1.00 580: return 0;
-1.00 581:}
-1.00 582:
 0.78 583:int getCost(int cardNumber)
-1.00 584:{
 0.78 585:  switch( cardNumber )
-1.00 586:   {
-1.00 587:   case curse:
 1.00 588:     return 0;
-1.00 589:   case estate:
 1.00 590:     return 2;
-1.00 591:   case duchy:
 0.99 592:     return 5;
-1.00 593:   case province:
 0.94 594:     return 8;
-1.00 595:   case copper:
 0.49 596:     return 0;
-1.00 597:   case silver:
 0.70 598:     return 3;
-1.00 599:   case gold:
 0.76 600:     return 6;
-1.00 601:   case adventurer:
 0.96 602:     return 6;
-1.00 603:   case council_room:
 1.00 604:     return 5;
-1.00 605:   case feast:
 1.00 606:     return 4;
-1.00 607:   case gardens:
-1.00 608:     return 4;
-1.00 609:   case mine:
 1.00 610:     return 5;
-1.00 611:   case remodel:
 1.00 612:     return 4;
-1.00 613:   case smithy:
 1.00 614:     return 4;
-1.00 615:   case village:
 1.00 616:     return 3;
-1.00 617:   case baron:
 0.00 618:     return 4;
-1.00 619:   case great_hall:
-1.00 620:     return 3;
-1.00 621:   case minion:
 0.00 622:     return 5;
-1.00 623:   case steward:
 1.00 624:     return 3;
-1.00 625:   case tribute:
 0.99 626:     return 5;
-1.00 627:   case ambassador:
 1.00 628:     return 3;
-1.00 629:   case cutpurse:
-1.00 630:     return 4;
-1.00 631:   case embargo:
 1.00 632:     return 2;
-1.00 633:   case outpost:
 1.00 634:     return 5;
-1.00 635:   case salvager:
 0.99 636:     return 4;
-1.00 637:   case sea_hag:
 0.00 638:     return 4;
-1.00 639:   case treasure_map:
 1.00 640:     return 4;
-1.00 641:   }
-1.00 642:
-1.00 643: return -1;
-1.00 644:}
-1.00 645:
 0.87 646:int cardEffect(int card, int choice1, int choice2, int choice3, struct gameState *state, int handPos, int *bonus)
-1.00 647:{
-1.00 648: int i;
-1.00 649: int j;
-1.00 650: int k;
```

```
-1.00 651:  int x;
-1.00 652:  int index;
 0.87 653:  int currentPlayer = whoseTurn(state);
 0.87 654:  int nextPlayer = currentPlayer + 1;
-1.00 655:
 0.87 656:  int tributeRevealedCards[2] = {-1, -1};
-1.00 657:  int temphand[MAX_HAND];// moved above the if statement
 0.87 658:  int drawntreasure=0;
-1.00 659:  int cardDrawn;
 0.87 660:  int z = 0;// this is the counter for the temp hand
 0.87 661:  if (nextPlayer > (state->numPlayers - 1)){
 0.85 662:    nextPlayer = 0;
-1.00 663:  }
-1.00 664:
-1.00 665:
-1.00 666:  //uses switch to select card and perform actions
 0.87 667:  switch( card )
-1.00 668:    {
-1.00 669:    case adventurer:
-1.00 670:      return adventurerEffect(state, handPos);
-1.00 671:
-1.00 672:    case council_room:
-1.00 673:      //+4 Cards
-1.00 674:      for (i = 0; i < 4; i++)
-1.00 675:                                                                      {
-1.00 676:                                                                        drawCard(currentPlayer, state);
-1.00 677:                                                                      }
-1.00 678:
-1.00 679:      //+1 Buy
-1.00 680:      state->numBuys++;
-1.00 681:
-1.00 682:      //Each other player draws a card
-1.00 683:      for (i = 0; i < state->numPlayers; i++)
-1.00 684:                                                                      {
-1.00 685:                                                                        if ( i != currentPlayer )
-1.00 686:                                                                          {
-1.00 687:                                                                            drawCard(i, state);
-1.00 688:                                                                          }
-1.00 689:                                                                      }
-1.00 690:
-1.00 691:      //put played card in played card pile
-1.00 692:      discardCard(handPos, currentPlayer, state, 0);
-1.00 693:
-1.00 694:      return 0;
-1.00 695:
-1.00 696:    case feast:
-1.00 697:      return feastEffect(choice1, state, handPos);
-1.00 698:
-1.00 699:    case gardens:
-1.00 700:      return -1;
-1.00 701:
-1.00 702:    case mine:
 0.87 703:      return mineEffect(choice1, choice2, state, handPos);
-1.00 704:
-1.00 705:    case remodel:
-1.00 706:      j = state->hand[currentPlayer][choice1];  //store card we will trash
-1.00 707:
-1.00 708:      if ( (getCost(state->hand[currentPlayer][choice1]) + 2) > getCost(choice2) )
-1.00 709:                                                                      {
-1.00 710:                                                                        return -1;
-1.00 711:                                                                      }
-1.00 712:
-1.00 713:      gainCard(choice2, state, 0, currentPlayer);
-1.00 714:
-1.00 715:      //discard card from hand
-1.00 716:      discardCard(handPos, currentPlayer, state, 0);
-1.00 717:
-1.00 718:      //discard trashed card
-1.00 719:      for (i = 0; i < state->handCount[currentPlayer]; i++)
-1.00 720:                                                                      {
-1.00 721:                                                                        if (state->hand[currentPlayer][i] == j)
-1.00 722:                                                                          {
-1.00 723:                                                                            discardCard(i, currentPlayer, state, 0);
-1.00 724:                                                                            break;
-1.00 725:                                                                          }
-1.00 726:                                                                      }
-1.00 727:
-1.00 728:
-1.00 729:      return 0;
-1.00 730:
-1.00 731:    case smithy:
-1.00 732:      //+3 Cards
```

```
-1.00  733:     return smithyEffect(state, handPos);
-1.00  734:
-1.00  735:   case village:
-1.00  736:     //+1 Card
-1.00  737:     drawCard(currentPlayer, state);
-1.00  738:
-1.00  739:     //+2 Actions
-1.00  740:     state->numActions = state->numActions + 2;
-1.00  741:
-1.00  742:     //discard played card from hand
-1.00  743:     discardCard(handPos, currentPlayer, state, 0);
-1.00  744:     return 0;
-1.00  745:
-1.00  746:   case baron:
-1.00  747:     state->numBuys++;//Increase buys by 1!
-1.00  748:     if (choice1 > 0){//Boolean true or going to discard an estate
-1.00  749:                                                                   int p = 0;//Iterator for hand!
-1.00  750:                                                                   int card_not_discarded = 1;//Flag for discard set!
-1.00  751:                                                                   while(card_not_discarded){
-1.00  752:                                                                     if (state->hand[currentPlayer][p] == estate){//Found an estate card!
-1.00  753:                                                                       state->coins += 4;//Add 4 coins to the amount of coins
-1.00  754:                                                                       state->discard[currentPlayer][state->discardCount[currentPlayer]] = state->hand[currentPlayer][p];
-1.00  755:                                                                       state->discardCount[currentPlayer]++;
-1.00  756:                                                                       for (;p < state->handCount[currentPlayer]; p++){
-1.00  757:                                                                         state->hand[currentPlayer][p] = state->hand[currentPlayer][p+1];
-1.00  758:                                                                       }
-1.00  759:                                                                       state->hand[currentPlayer][state->handCount[currentPlayer]] = -1;
-1.00  760:                                                                       state->handCount[currentPlayer]--;
-1.00  761:                                                                       card_not_discarded = 0;//Exit the loop
-1.00  762:                                                                     }
-1.00  763:                                                                     else if (p > state->handCount[currentPlayer]){
-1.00  764:                                                                       if(DEBUG) {
-1.00  765:                                                                         printf("No estate cards in your hand, invalid choice\n");
-1.00  766:                                                                         printf("Must gain an estate if there are any\n");
-1.00  767:                                                                       }
-1.00  768:                                                                       if (supplyCount(estate, state) > 0){
-1.00  769:                                                                         gainCard(estate, state, 0, currentPlayer);
-1.00  770:                                                                         state->supplyCount[estate]--;//Decrement estates
-1.00  771:                                                                         if (supplyCount(estate, state) == 0){
-1.00  772:                                                                                                                                                   isGameOver(state);
-1.00  773:                                                                         }
-1.00  774:                                                                       }
-1.00  775:                                                                       card_not_discarded = 0;//Exit the loop
-1.00  776:                                                                     }
-1.00  777:
-1.00  778:                                                                     else{
-1.00  779:                                                                       p++;//Next card
-1.00  780:                                                                     }
-1.00  781:                                                                   }
-1.00  782:     }
-1.00  783:
-1.00  784:     else{
-1.00  785:                                                                   if (supplyCount(estate, state) > 0){
-1.00  786:                                                                     gainCard(estate, state, 0, currentPlayer);//Gain an estate
-1.00  787:                                                                     state->supplyCount[estate]--;//Decrement Estates
-1.00  788:                                                                     if (supplyCount(estate, state) == 0){
-1.00  789:                                                                       isGameOver(state);
-1.00  790:                                                                     }
-1.00  791:                                                                   }
-1.00  792:     }
-1.00  793:
-1.00  794:
-1.00  795:     return 0;
-1.00  796:
-1.00  797:   case great_hall:
-1.00  798:     //+1 Card
-1.00  799:     drawCard(currentPlayer, state);
-1.00  800:
-1.00  801:     //+1 Actions
-1.00  802:     state->numActions++;
-1.00  803:
-1.00  804:     //discard card from hand
-1.00  805:     discardCard(handPos, currentPlayer, state, 0);
-1.00  806:     return 0;
-1.00  807:
-1.00  808:   case minion:
-1.00  809:     return minionEffect(choice1, choice2, state, handPos);
-1.00  810:
-1.00  811:   case steward:
-1.00  812:     if (choice1 == 1)
-1.00  813:                                                                   {
-1.00  814:                                                                     //+2 cards
```

```
-1.00           drawCard(currentPlayer, state);
-1.00           drawCard(currentPlayer, state);
-1.00         }
-1.00     else if (choice1 == 2)
-1.00         {
-1.00           //+2 coins
-1.00           state->coins = state->coins + 2;
-1.00         }
-1.00     else
-1.00         {
-1.00           //trash 2 cards in hand
-1.00           discardCard(choice2, currentPlayer, state, 1);
-1.00           discardCard(choice3, currentPlayer, state, 1);
-1.00         }
-1.00
-1.00     //discard card from hand
-1.00     discardCard(handPos, currentPlayer, state, 0);
-1.00     return 0;
-1.00
-1.00   case tribute:
-1.00     if ((state->discardCount[nextPlayer] + state->deckCount[nextPlayer]) <= 1){
-1.00         if (state->deckCount[nextPlayer] > 0){
-1.00           tributeRevealedCards[0] = state->deck[nextPlayer][state->deckCount[nextPlayer]-1];
-1.00           state->deckCount[nextPlayer]--;
-1.00         }
-1.00         else if (state->discardCount[nextPlayer] > 0){
-1.00           tributeRevealedCards[0] = state->discard[nextPlayer][state->discardCount[nextPlayer]-1];
-1.00           state->discardCount[nextPlayer]--;
-1.00         }
-1.00         else{
-1.00           //No Card to Reveal
-1.00           if (DEBUG){
-1.00             printf("No cards to reveal\n");
-1.00           }
-1.00         }
-1.00     }
-1.00
-1.00     else{
-1.00         if (state->deckCount[nextPlayer] == 0){
-1.00           for (i = 0; i < state->discardCount[nextPlayer]; i++){
-1.00             state->deck[nextPlayer][i] = state->discard[nextPlayer][i];//Move to deck
-1.00             state->deckCount[nextPlayer]++;
-1.00             state->discard[nextPlayer][i] = -1;
-1.00             state->discardCount[nextPlayer]--;
-1.00           }
-1.00
-1.00           shuffle(nextPlayer,state);//Shuffle the deck
-1.00         }
-1.00         tributeRevealedCards[0] = state->deck[nextPlayer][state->deckCount[nextPlayer]-1];
-1.00         state->deck[nextPlayer][state->deckCount[nextPlayer]--] = -1;
-1.00         state->deckCount[nextPlayer]--;
-1.00         tributeRevealedCards[1] = state->deck[nextPlayer][state->deckCount[nextPlayer]-1];
-1.00         state->deck[nextPlayer][state->deckCount[nextPlayer]--] = -1;
-1.00         state->deckCount[nextPlayer]--;
-1.00     }
-1.00
-1.00     if (tributeRevealedCards[0] == tributeRevealedCards[1]){//If we have a duplicate card, just drop one
-1.00         state->playedCards[state->playedCardCount] = tributeRevealedCards[1];
-1.00         state->playedCardCount++;
-1.00         tributeRevealedCards[1] = -1;
-1.00     }
-1.00
-1.00     for (i = 0; i <= 2; i ++){
-1.00         if (tributeRevealedCards[i] == copper || tributeRevealedCards[i] == silver || tributeRevealedCards[i] == gold){//Treasure cards
-1.00           state->coins += 2;
-1.00         }
-1.00
-1.00         else if (tributeRevealedCards[i] == estate || tributeRevealedCards[i] == duchy || tributeRevealedCards[i] == province || tributeRevealedCards[i] == gardens || tributeRevealedCards[i] == great_hall){//Victory Card Found
-1.00           drawCard(currentPlayer, state);
-1.00           drawCard(currentPlayer, state);
-1.00         }
-1.00         else{//Action Card
-1.00           state->numActions = state->numActions + 2;
-1.00         }
-1.00     }
-1.00
-1.00     return 0;
-1.00
-1.00   case ambassador:
-1.00     j = 0;                    //used to check if player has enough cards to discard
-1.00
-1.00     if (choice2 > 2 || choice2 < 0)
```

```
            {
              return -1;
            }

    if (choice1 == handPos)
            {
              return -1;
            }

    for (i = 0; i < state->handCount[currentPlayer]; i++)
            {
              if (i != handPos && i == state->hand[currentPlayer][choice1] && i != choice1)
                {
                  j++;
                }
            }
    if (j < choice2)
            {
              return -1;
            }

    if (DEBUG)
            printf("Player %d reveals card number: %d\n", currentPlayer, state->hand[currentPlayer][choice1]);

    //increase supply count for choosen card by amount being discarded
    state->supplyCount[state->hand[currentPlayer][choice1]] += choice2;

    //each other player gains a copy of revealed card
    for (i = 0; i < state->numPlayers; i++)
            {
              if (i != currentPlayer)
                {
                  gainCard(state->hand[currentPlayer][choice1], state, 0, i);
                }
            }

    //discard played card from hand
    discardCard(handPos, currentPlayer, state, 0);

    //trash copies of cards returned to supply
    for (j = 0; j < choice2; j++)
            {
              for (i = 0; i < state->handCount[currentPlayer]; i++)
                {
                  if (state->hand[currentPlayer][i] == state->hand[currentPlayer][choice1])
                        {
                          discardCard(i, currentPlayer, state, 1);
                          break;
                        }
                }
            }

    return 0;

  case cutpurse:

    updateCoins(currentPlayer, state, 2);
    for (i = 0; i < state->numPlayers; i++)
            {
              if (i != currentPlayer)
                {
                  for (j = 0; j < state->handCount[i]; j++)
                        {
                          if (state->hand[i][j] == copper)
                            {
                              discardCard(j, i, state, 0);
                              break;
                            }
                          if (j == state->handCount[i])
                            {
                              for (k = 0; k < state->handCount[i]; k++)
                                    {
                                      if (DEBUG)
                                        printf("Player %d reveals card number %d\n", i, state->hand[i][k]);
                                    }
                              break;
                            }
                        }

                }

            }
```

```
-1.00
-1.00     //discard played card from hand
-1.00     discardCard(handPos, currentPlayer, state, 0);
-1.00
-1.00     return 0;
-1.00
-1.00
-1.00   case embargo:
-1.00     //+2 Coins
-1.00     state->coins = state->coins + 2;
-1.00
-1.00     //see if selected pile is in play
-1.00     if ( state->supplyCount[choice1] == -1 )
-1.00       {
-1.00         return -1;
-1.00       }
-1.00
-1.00     //add embargo token to selected supply pile
-1.00     state->embargoTokens[choice1]++;
-1.00
-1.00     //trash card
-1.00     discardCard(handPos, currentPlayer, state, 1);
-1.00     return 0;
-1.00
-1.00   case outpost:
-1.00     //set outpost flag
-1.00     state->outpostPlayed++;
-1.00
-1.00     //discard card
-1.00     discardCard(handPos, currentPlayer, state, 0);
-1.00     return 0;
-1.00
-1.00   case salvager:
-1.00     //+1 buy
-1.00     state->numBuys++;
-1.00
-1.00     if (choice1)
-1.00       {
-1.00         //gain coins equal to trashed card
-1.00         state->coins = state->coins + getCost( handCard(choice1, state) );
-1.00         //trash card
-1.00         discardCard(choice1, currentPlayer, state, 1);
-1.00       }
-1.00
-1.00     //discard card
-1.00     discardCard(handPos, currentPlayer, state, 0);
-1.00     return 0;
-1.00
-1.00   case sea_hag:
-1.00     for (i = 0; i < state->numPlayers; i++){
-1.00       if (i != currentPlayer){
-1.00         state->discard[i][state->discardCount[i]] = state->deck[i][state->deckCount[i]--];     state->deckCount[i]--;
-1.00         state->discardCount[i]++;
-1.00         state->deck[i][state->deckCount[i]--] = curse;//Top card now a curse
-1.00       }
-1.00     }
-1.00     return 0;
-1.00
-1.00   case treasure_map:
-1.00     //search hand for another treasure_map
-1.00     index = -1;
-1.00     for (i = 0; i < state->handCount[currentPlayer]; i++)
-1.00       {
-1.00         if (state->hand[currentPlayer][i] == treasure_map && i != handPos)
-1.00           {
-1.00             index = i;
-1.00             break;
-1.00           }
-1.00       }
-1.00     if (index > -1)
-1.00       {
-1.00         //trash both treasure cards
-1.00         discardCard(handPos, currentPlayer, state, 1);
-1.00         discardCard(index, currentPlayer, state, 1);
-1.00
-1.00         //gain 4 Gold cards
-1.00         for (i = 0; i < 4; i++)
-1.00           {
-1.00             gainCard(gold, state, 1, currentPlayer);
-1.00           }
-1.00
-1.00         //return success
```

```
-1.00  1061:                                                                    return 1;
-1.00  1062:                                                                  }
-1.00  1063:
-1.00  1064:     //no second treasure_map found in hand
-1.00  1065:     return -1;
-1.00  1066:   }
-1.00  1067:
-1.00  1068:  return -1;
-1.00  1069:}
-1.00  1070:
-1.00  1071:int adventurerEffect(struct gameState *state, int handPos) {
-1.00  1072:   int i, currentPlayer = whoseTurn(state);
-1.00  1073:   int temphand[MAX_HAND];
-1.00  1074:   int drawntreasure = 0;
-1.00  1075:   int cardDrawn;
-1.00  1076:   int topCard;
-1.00  1077:   int totDeck; // counter for total cards in deck
-1.00  1078:   int totDiscard; // counter for total cards in discard
-1.00  1079:   int y = 0; // counter for the shuffle
-1.00  1080:   int z = 0; // counter for temp hand
-1.00  1081:
-1.00  1082:   totDeck = state->deckCount[currentPlayer];
-1.00  1083:   totDiscard = state->discardCount[currentPlayer];
-1.00  1084:
-1.00  1085:   while((drawntreasure < 2) && ((totDeck + totDiscard) != 0)) {
-1.00  1086:     // if the deck is empty, we need to shuffle discard and add to deck
-1.00  1087:     if(totDeck < 1) {
-1.00  1088:       shuffle(currentPlayer, state);
-1.00  1089:     }
-1.00  1090:     drawCard(currentPlayer, state);
-1.00  1091:     // top card is most recently drawn card
-1.00  1092:     topCard = state->handCount[currentPlayer] - 1;
-1.00  1093:     cardDrawn = state->hand[currentPlayer][topCard];
-1.00  1094:     if(cardDrawn == copper || cardDrawn == silver || cardDrawn == gold) {
-1.00  1095:       drawntreasure++;
-1.00  1096:     }
-1.00  1097:     else {
-1.00  1098:       temphand[z] = cardDrawn;
-1.00  1099:       // remove the top card (the most recently drawn one)
-1.00  1100:       state->handCount[currentPlayer]--;
-1.00  1101:       z++;
-1.00  1102:     }
-1.00  1103:     totDeck = state->deckCount[currentPlayer];
-1.00  1104:     totDiscard = state->discardCount[currentPlayer];
-1.00  1105:   }
-1.00  1106:   // discard all non-treasure cards that have been drawn
-1.00  1107:   while((z - 1) >= 0) {
-1.00  1108:     topCard = state->discardCount[currentPlayer];
-1.00  1109:     state->discard[currentPlayer][topCard++] = temphand[z - 1];
-1.00  1110:     state->discardCount[currentPlayer] = topCard;
-1.00  1111:     z = z - 1;
-1.00  1112:   }
-1.00  1113:   // discard the adventurer card from hand
-1.00  1114:   discardCard(handPos, currentPlayer, state, 0);
-1.00  1115:
-1.00  1116:   return 0;
-1.00  1117:}
-1.00  1118:
-1.00  1119:// smithy allows a player to draw 3 cards
-1.00  1120:int smithyEffect(struct gameState *state, int handPos) {
-1.00  1121:   int currentPlayer = whoseTurn(state);
-1.00  1122:   int i;
-1.00  1123:
-1.00  1124:   for(i = 0; i < 3; i++) {
-1.00  1125:     drawCard(currentPlayer, state);
-1.00  1126:   }
-1.00  1127:   return 0;
-1.00  1128:}
-1.00  1129:
-1.00  1130:// feast allows you to gain a card with a cost up to 5
-1.00  1131:int feastEffect(int choice, struct gameState *state, int handPos) {
-1.00  1132:   int currentPlayer = whoseTurn(state);
-1.00  1133:   int cardCountHand;
-1.00  1134:   int cardCountDeck;
-1.00  1135:   int cardCountDiscard;
-1.00  1136:   int cardCountTotal;
-1.00  1137:
-1.00  1138:   // Update coins for buy
-1.00  1139:   updateCoins(currentPlayer, state, 5);
-1.00  1140:   // Buy one card
-1.00  1141:   if(supplyCount(choice, state) <= 0) {
-1.00  1142:     if(DEBUG) {
```

```
-1.00  1143:       printf("Cards Left: %d\n", supplyCount(choice, state));
-1.00  1144:     }
-1.00  1145:     return -1;
-1.00  1146:   }
-1.00  1147:   else if(state->coins < getCost(choice)) {
-1.00  1148:     if(DEBUG) {
-1.00  1149:       printf("Coins: %d < %d\n", state->coins, getCost(choice));
-1.00  1150:     }
-1.00  1151:     return -1;
-1.00  1152:   }
-1.00  1153:   else {
-1.00  1154:     if(DEBUG) {
-1.00  1155:       cardCountHand = state->handCount[currentPlayer];
-1.00  1156:       cardCountDeck = state->deckCount[currentPlayer];
-1.00  1157:       cardCountDiscard = state->discardCount[currentPlayer];
-1.00  1158:       cardCountTotal = cardCountHand + cardCountDeck + cardCountDiscard;
-1.00  1159:       printf("Deck Count: %d\n", cardCountTotal);
-1.00  1160:     }
-1.00  1161:     // discard and trash the feast card
-1.00  1162:     discardCard(handPos, currentPlayer, state, 1);
-1.00  1163:
-1.00  1164:     gainCard(choice, state, 0, currentPlayer); // Gain the card
-1.00  1165:
-1.00  1166:     if(DEBUG) {
-1.00  1167:       cardCountHand = state->handCount[currentPlayer];
-1.00  1168:       cardCountDeck = state->deckCount[currentPlayer];
-1.00  1169:       cardCountDiscard = state->discardCount[currentPlayer];
-1.00  1170:       cardCountTotal = cardCountHand + cardCountDeck + cardCountDiscard;
-1.00  1171:       printf("Deck Count: %d\n", cardCountTotal);
-1.00  1172:     }
-1.00  1173:   }
-1.00  1174:   return 0;
-1.00  1175:}
-1.00  1176:
-1.00  1177:// Minion is a +1 action card
-1.00  1178:int minionEffect(int choice1, int choice2, struct gameState *state, int handPos)
-1.00  1179:{
-1.00  1180:   int i;
-1.00  1181:   int j;
-1.00  1182:   int currentPlayer = whoseTurn(state);
-1.00  1183:
-1.00  1184:   state->numActions++;
-1.00  1185:
-1.00  1186:   // discard card from hand
-1.00  1187:   discardCard(handPos, currentPlayer, state, 0);
-1.00  1188:
-1.00  1189:   // if the player chooses the +2 coins option
-1.00  1190:   if(choice1) {
-1.00  1191:     state->coins = state->coins + 2;
-1.00  1192:   }
-1.00  1193:
-1.00  1194:   // if the players chooses to discard hand, redraw 4, and other players with
-1.00  1195:   // 5+ cards discard hand and draw 4
-1.00  1196:   else if(choice2) {
-1.00  1197:     // discard hand
-1.00  1198:     while(numHandCards(state) > 0) {
-1.00  1199:       discardCard(handPos, currentPlayer, state, 0);
-1.00  1200:     }
-1.00  1201:     // draw 4
-1.00  1202:     for(i = 0; i < 4; i++) {
-1.00  1203:       drawCard(currentPlayer, state);
-1.00  1204:     }
-1.00  1205:     // other players discard hand redraw if hand size > 4
-1.00  1206:     for(i = 0; i < state->numPlayers; i++) {
-1.00  1207:       if((i != currentPlayer) && ((state->handCount[i]) > 4)) {
-1.00  1208:         // discard hand
-1.00  1209:         while(state->handCount[i] > 0) {
-1.00  1210:           discardCard(handPos, i, state, 0);
-1.00  1211:         }
-1.00  1212:         // draw 4
-1.00  1213:         for(j = 0; j < 4; j++) {
-1.00  1214:           drawCard(i, state);
-1.00  1215:         }
-1.00  1216:       }
-1.00  1217:     }
-1.00  1218:   }
-1.00  1219:   return 0;
-1.00  1220:}
-1.00  1221:
 0.87  1222:int mineEffect(int choice1, int choice2, struct gameState *state, int handPos)
-1.00  1223:{
-1.00  1224:   int i;
```

```
 0.87 1225:   int currentPlayer = whoseTurn(state);
 0.87 1226:   int j = state->hand[currentPlayer][choice1]; // store card we will trash
-1.00 1227:
 0.87 1228:   if((state->hand[currentPlayer][choice1] < copper) ||
 0.88 1229:     (state->hand[currentPlayer][choice1] > gold)) {
 0.00 1230:     return -1;
-1.00 1231:   }
 0.94 1232:   if((choice2 > treasure_map) || (choice2 < curse)) {
 0.00 1233:     return -1;
-1.00 1234:   }
 0.95 1235:   if((getCost(state->hand[currentPlayer][choice1]) + 3) < getCost(choice2)) {
 0.00 1236:     return -1;
-1.00 1237:   }
 0.96 1238:   gainCard(choice2, state, 2, currentPlayer);
-1.00 1239:
-1.00 1240:   // discard card from hand
 0.96 1241:   discardCard(handPos, currentPlayer, state, 0);
-1.00 1242:
-1.00 1243:   // discard trashed card
 0.96 1244:   for(i = 0; i < (state->handCount[currentPlayer]); i++) {
 0.96 1245:     if(state->hand[currentPlayer][i] == j) {
 0.96 1246:       discardCard(i, currentPlayer, state, 1);
 0.96 1247:       break;
-1.00 1248:     }
-1.00 1249:   }
 0.96 1250:   return 0;
-1.00 1251:}
-1.00 1252:
 0.96 1253:int discardCard(int handPos, int currentPlayer, struct gameState *state, int trashFlag)
-1.00 1254:{
-1.00 1255:
-1.00 1256:  //if card is not trashed, added to Played pile
 0.96 1257:  if (trashFlag < 1)
-1.00 1258:   {
-1.00 1259:     //add card to played pile
 0.96 1260:     state->playedCards[state->playedCardCount] = state->hand[currentPlayer][handPos];
 0.96 1261:     state->playedCardCount++;
-1.00 1262:   }
-1.00 1263:
-1.00 1264:  //set played card to -1
 0.96 1265:  state->hand[currentPlayer][handPos] = -1;
-1.00 1266:
-1.00 1267:  //remove card from player's hand
 0.96 1268:  if ( handPos == (state->handCount[currentPlayer] - 1) )                                    //last card in hand array is played
-1.00 1269:   {
-1.00 1270:     //reduce number of cards in hand
 0.98 1271:     state->handCount[currentPlayer]--;
-1.00 1272:   }
 0.96 1273:  else if ( state->handCount[currentPlayer] == 1 ) //only one card in hand
-1.00 1274:   {
-1.00 1275:     //reduce number of cards in hand
-1.00 1276:     state->handCount[currentPlayer]--;
-1.00 1277:   }
-1.00 1278:  else
-1.00 1279:   {
-1.00 1280:     //replace discarded card with last card in hand
 0.96 1281:     state->hand[currentPlayer][handPos] = state->hand[currentPlayer][ (state->handCount[currentPlayer] - 1)];
-1.00 1282:     //set last card to -1
 0.96 1283:     state->hand[currentPlayer][state->handCount[currentPlayer] - 1] = -1;
-1.00 1284:     //reduce number of cards in hand
 0.96 1285:     state->handCount[currentPlayer]--;
-1.00 1286:   }
-1.00 1287:
 0.96 1288:  return 0;
-1.00 1289:}
-1.00 1290:
 0.96 1291:int gainCard(int supplyPos, struct gameState *state, int toFlag, int player)
-1.00 1292:{
-1.00 1293:  //Note: supplyPos is enum of choosen card
-1.00 1294:
-1.00 1295:  //check if supply pile is empty (0) or card is not used in game (-1)
 0.96 1296:  if ( supplyCount(supplyPos, state) < 1 )
-1.00 1297:   {
 1.00 1298:     return -1;
-1.00 1299:   }
-1.00 1300:
-1.00 1301:  //added card for [whoseTurn] current player:
-1.00 1302:  // toFlag = 0 : add to discard
-1.00 1303:  // toFlag = 1 : add to deck
-1.00 1304:  // toFlag = 2 : add to hand
-1.00 1305:
 0.95 1306:  if (toFlag == 1)
```

```
-1.00 1307:   {
-1.00 1308:     state->deck[ player ][ state->deckCount[player] ] = supplyPos;
-1.00 1309:     state->deckCount[player]++;
-1.00 1310:   }
 0.95 1311: else if (toFlag == 2)
-1.00 1312:   {
 0.95 1313:     state->hand[ player ][ state->handCount[player] ] = supplyPos;
 0.95 1314:     state->handCount[player]++;
-1.00 1315:   }
-1.00 1316: else
-1.00 1317:   {
-1.00 1318:     state->discard[player][ state->discardCount[player] ] = supplyPos;
-1.00 1319:     state->discardCount[player]++;
-1.00 1320:   }
-1.00 1321:
-1.00 1322: //decrease number in supply pile
 0.95 1323: state->supplyCount[supplyPos]--;
-1.00 1324:
 0.95 1325: return 0;
-1.00 1326:}
-1.00 1327:
 0.96 1328:int updateCoins(int player, struct gameState *state, int bonus)
-1.00 1329:{
-1.00 1330: int i;
-1.00 1331:
-1.00 1332: //reset coin count
 0.96 1333: state->coins = 0;
-1.00 1334:
-1.00 1335: //add coins for each Treasure card in player's hand
 0.96 1336: for (i = 0; i < state->handCount[player]; i++)
-1.00 1337:   {
 0.96 1338:     if (state->hand[player][i] == copper)
-1.00 1339:                                                 {
 0.94 1340:                                                   state->coins += 1;
-1.00 1341:                                                 }
 0.96 1342:     else if (state->hand[player][i] == silver)
-1.00 1343:                                                 {
 0.95 1344:                                                   state->coins += 2;
-1.00 1345:                                                 }
 0.96 1346:     else if (state->hand[player][i] == gold)
-1.00 1347:                                                 {
 0.95 1348:                                                   state->coins += 3;
-1.00 1349:                                                 }
-1.00 1350:   }
-1.00 1351:
-1.00 1352: //add bonus
 0.96 1353: state->coins += bonus;
-1.00 1354:
 0.96 1355: return 0;
-1.00 1356:}
-1.00 1357:
-1.00 1358:
-1.00 1359://end of dominion.c
```