Will Olsen

olsenw@onid.oregonstate.edu

CS 362 Final

Testing my copy of dominion was an interesting and thoughtful experience. I am documenting how I tested my copy of dominion, as I followed a similar strategy/pattern for my classmates' dominions.

I think the most helpful test to run has been the dominion game player written for assignment number four. The reason being that it provided a log of the game that I could read and analyze. This is what led me to discover that my scorefor function in dominion.c was returning incorrect scores. The score for any particular player could fluctuate from turn to turn, without the purchase of victory cards.

Running the dominion game player also provided an excellent coverage output, by running through many of the lines of code, revealing some spots with dead code. An example would be the function on line 21. I included the code coverage of one run of the domino game player, I will not include other coverages as it is long.

After running the dominion game player I ran some individual unit tests which allowed a more in depth look at the behavior of some of the cards. This proved to be effective to help find some specific issues within certain functions. Such as the scorefor issue revealed by the dominion game player.

My copy of dominion.c I think is unreliable at best, and highly error prone at worst. While a typical simple game of dominion is likely to play with little issue, excepting incorrect score accounting, certain cases and conditions can cause the code to fail completely.

Testing huangma code I believe it has the same basic reliably as mine. Which is to say little to none. It has many of the same bugs as mine does, such as inaccurate score accounting.

To test his code I simply ran the testers I wrote for my own code. Many of the bugs this tests found in my code were found here. An example was the random tester used to test the adventure card. Several times the tester stopped responding, in what turned out to be an infinite loop caused when there was less that two treasures in the players' deck.

Testing brooksc code is also fairly unreliable. The same scoring bug and adventure infinite loop also exist.

Like I did huangma I ran the testers I had on their code. One test I had testrandomcard that tested council room found a bug. All of the tests would fail because the other players were gaining an extra card. The same infinite loop in adventure was also found with testadventurecard, which happened when there was less than two treasure in the dominion deck.

GCOV of my dominion.c after one run of the dominion game player:

```
    -:    0:Source:dominion.c
    -:    0:Graph:dominion.gcno
    -:    0:Data:dominion.gcda
    -:    0:Runs:1
    -:    0:Programs:1
    -:    1:#include "dominion.h"
    -:    2:#include "dominion_helpers.h"
    -:    3:#include "rngs.h"
    -:    4:#include <stdio.h>
    -:    5:#include <math.h>
    -:    6:#include <stdlib.h>
    -:    7:
 3219:    8:int compare(const void* a, const void* b) {
 3219:    9:  if (*(int*)a > *(int*)b)
 1545:   10:    return 1;
 1674:   11:  if (*(int*)a < *(int*)b)
  632:   12:    return -1;
 1042:   13:  return 0;
    -:   14:}
    -:   15:
#####:   16:struct gameState* newGame() {
#####:   17:  struct gameState* g = malloc(sizeof(struct gameState));
#####:   18:  return g;
    -:   19:}
    -:   20:
#####:   21:int* kingdomCards(int k1, int k2, int k3, int k4, int k5, int k6, int k7,
    -:   22:                  int k8, int k9, int k10) {
#####:   23:  int* k = malloc(10 * sizeof(int));
```

```
#####:   24:  k[0] = k1;
#####:   25:  k[1] = k2;
#####:   26:  k[2] = k3;
#####:   27:  k[3] = k4;
#####:   28:  k[4] = k5;
#####:   29:  k[5] = k6;
#####:   30:  k[6] = k7;
#####:   31:  k[7] = k8;
#####:   32:  k[8] = k9;
#####:   33:  k[9] = k10;
#####:   34:  return k;
   -:   35:}
   -:   36:
   1:   37:int initializeGame(int numPlayers, int kingdomCards[10], int randomSeed,
   -:   38:                  struct gameState *state) {
   -:   39:
   -:   40:  int i;
   -:   41:  int j;
   -:   42:  int it;
   -:   43:  //set up random number generator
   1:   44:  SelectStream(1);
   1:   45:  PutSeed((long)randomSeed);
   -:   46:
   -:   47:  //check number of players
   1:   48:  if (numPlayers > MAX_PLAYERS || numPlayers < 2)
   -:   49:    {
#####:   50:      return -1;
   -:   51:    }
   -:   52:
```

```
    -:   53:  //set number of players
    1:   54:  state->numPlayers = numPlayers;
    -:   55:
    -:   56:  //check selected kingdom cards are different
   11:   57:  for (i = 0; i < 10; i++)
    -:   58:    {
  110:   59:      for (j = 0; j < 10; j++)
    -:   60:       {
  100:   61:         if (j != i && kingdomCards[j] == kingdomCards[i])
    -:   62:           {
#####:   63:             return -1;
    -:   64:           }
    -:   65:       }
    -:   66:    }
    -:   67:
    -:   68:
    -:   69:  //initialize supply
    -:   70:  ////////////////////////////////
    -:   71:
    -:   72:  //set number of Curse cards
    1:   73:  if (numPlayers == 2)
    -:   74:    {
    1:   75:      state->supplyCount[curse] = 10;
    -:   76:    }
#####:   77:  else if (numPlayers == 3)
    -:   78:    {
#####:   79:      state->supplyCount[curse] = 20;
    -:   80:    }
    -:   81:  else
```

```
   -:  82:    {
#####:  83:      state->supplyCount[curse] = 30;
   -:  84:    }
   -:  85:
   -:  86:  //set number of Victory cards
   1:  87:  if (numPlayers == 2)
   -:  88:    {
   1:  89:      state->supplyCount[estate] = 8;
   1:  90:      state->supplyCount[duchy] = 8;
   1:  91:      state->supplyCount[province] = 8;
   -:  92:    }
   -:  93:  else
   -:  94:    {
#####:  95:      state->supplyCount[estate] = 12;
#####:  96:      state->supplyCount[duchy] = 12;
#####:  97:      state->supplyCount[province] = 12;
   -:  98:    }
   -:  99:
   -:  100:  //set number of Treasure cards
   1:  101:  state->supplyCount[copper] = 60 - (7 * numPlayers);
   1:  102:  state->supplyCount[silver] = 40;
   1:  103:  state->supplyCount[gold] = 30;
   -:  104:
   -:  105:  //set number of Kingdom cards
  21:  106:  for (i = adventurer; i <= treasure_map; i++)            //loop all cards
   -:  107:    {
 165:  108:      for (j = 0; j < 10; j++)                            //loop chosen cards
   -:  109:        {
 155:  110:        if (kingdomCards[j] == i)
```

```
   -:  111:          {
   -:  112:            //check if card is a 'Victory' Kingdom card
  10:  113:          if (kingdomCards[j] == great_hall || kingdomCards[j] == gardens)
   -:  114:                {
#####:  115:                  if (numPlayers == 2){
#####:  116:                    state->supplyCount[i] = 8;
   -:  117:                  }
#####:  118:                  else{ state->supplyCount[i] = 12; }
   -:  119:                }
   -:  120:            else
   -:  121:                {
  10:  122:                  state->supplyCount[i] = 10;
   -:  123:                }
  10:  124:          break;
   -:  125:          }
   -:  126:         else    //card is not in the set choosen for the game
   -:  127:          {
 145:  128:          state->supplyCount[i] = -1;
   -:  129:          }
   -:  130:       }
   -:  131:
   -:  132:    }
   -:  133:
   -:  134:  ////////////////////////
   -:  135:  //supply intilization complete
   -:  136:
   -:  137:  //set player decks
   3:  138:  for (i = 0; i < numPlayers; i++)
   -:  139:    {
```

```
    2:  140:     state->deckCount[i] = 0;
    8:  141:     for (j = 0; j < 3; j++)
    -:  142:       {
    6:  143:         state->deck[i][j] = estate;
    6:  144:         state->deckCount[i]++;
    -:  145:       }
   16:  146:     for (j = 3; j < 10; j++)
    -:  147:       {
   14:  148:       state->deck[i][j] = copper;
   14:  149:       state->deckCount[i]++;
    -:  150:       }
    -:  151:   }
    -:  152:
    -:  153:  //shuffle player decks
    3:  154:  for (i = 0; i < numPlayers; i++)
    -:  155:   {
    2:  156:     if ( shuffle(i, state) < 0 )
    -:  157:       {
#####:  158:     return -1;
    -:  159:       }
    -:  160:   }
    -:  161:
    -:  162:  //draw player hands
    3:  163:  for (i = 0; i < numPlayers; i++)
    -:  164:   {
    -:  165:     //initialize hand size to zero
    2:  166:     state->handCount[i] = 0;
    2:  167:     state->discardCount[i] = 0;
    -:  168:     //draw 5 cards
```

```
-:  169:     // for (j = 0; j < 5; j++)
-:  170:     //           {
-:  171:     //             drawCard(i, state);
-:  172:     //           }
-:  173:   }
-:  174:
-:  175:  //set embargo tokens to 0 for all supply piles
28:  176:  for (i = 0; i <= treasure_map; i++)
-:  177:   {
27:  178:     state->embargoTokens[i] = 0;
-:  179:   }
-:  180:
-:  181:  //initialize first player's turn
 1:  182:  state->outpostPlayed = 0;
 1:  183:  state->phase = 0;
 1:  184:  state->numActions = 1;
 1:  185:  state->numBuys = 1;
 1:  186:  state->playedCardCount = 0;
 1:  187:  state->whoseTurn = 0;
 1:  188:  state->handCount[state->whoseTurn] = 0;
-:  189:  //int it; move to top
-:  190:
-:  191:  //Moved draw cards to here, only drawing at the start of a turn
 6:  192:  for (it = 0; it < 5; it++){
 5:  193:    drawCard(state->whoseTurn, state);
-:  194:  }
-:  195:
 1:  196:  updateCoins(state->whoseTurn, state, 0);
-:  197:
```

```
    1:  198:  return 0;
    -:  199:}
    -:  200:
   44:  201:int shuffle(int player, struct gameState *state) {
    -:  202:
    -:  203:
    -:  204:  int newDeck[MAX_DECK];
   44:  205:  int newDeckPos = 0;
    -:  206:  int card;
    -:  207:  int i;
    -:  208:
   44:  209:  if (state->deckCount[player] < 1)
#####:  210:    return -1;
   44:  211:  qsort ((void*)(state->deck[player]), state->deckCount[player], sizeof(int), compare);
    -:  212:  /* SORT CARDS IN DECK TO ENSURE DETERMINISM! */
    -:  213:
 1042:  214:  while (state->deckCount[player] > 0) {
  954:  215:    card = floor(Random() * state->deckCount[player]);
  954:  216:    newDeck[newDeckPos] = state->deck[player][card];
  954:  217:    newDeckPos++;
 7329:  218:    for (i = card; i < state->deckCount[player]-1; i++) {
 6375:  219:      state->deck[player][i] = state->deck[player][i+1];
    -:  220:    }
  954:  221:    state->deckCount[player]--;
    -:  222:  }
  998:  223:  for (i = 0; i < newDeckPos; i++) {
  954:  224:    state->deck[player][i] = newDeck[i];
  954:  225:    state->deckCount[player]++;
    -:  226:  }
```

```
  -:  227:
 44:  228:  return 0;
  -:  229:}
  -:  230:
571:  231:int playCard(int handPos, int choice1, int choice2, int choice3, struct gameState *state)
  -:  232:{
  -:  233:  int card;
571:  234:  int coin_bonus = 0;                    //tracks coins gain from actions
  -:  235:
  -:  236:  //check if it is the right phase
571:  237:  if (state->phase != 0)
  -:  238:    {
#####:  239:      return -1;
  -:  240:    }
  -:  241:
  -:  242:  //check if player has enough actions
571:  243:  if ( state->numActions < 1 )
  -:  244:    {
 97:  245:      return -1;
  -:  246:    }
  -:  247:
  -:  248:  //get card played
474:  249:  card = handCard(handPos, state);
  -:  250:
  -:  251:  //check if selected card is an action
474:  252:  if ( card < adventurer || card > treasure_map )
  -:  253:    {
387:  254:      return -1;
  -:  255:    }
```

```
    -:  256:
    -:  257:  //play card
   87:  258:  if ( cardEffect(card, choice1, choice2, choice3, state, handPos, &coin_bonus) < 0 )
    -:  259:    {
   48:  260:      return -1;
    -:  261:    }
    -:  262:
    -:  263:  //reduce number of actions
   39:  264:  state->numActions--;
    -:  265:
    -:  266:  //update coins (Treasure cards may be added with card draws)
   39:  267:  updateCoins(state->whoseTurn, state, coin_bonus);
    -:  268:
   39:  269:  return 0;
    -:  270:}
    -:  271:
  171:  272:int buyCard(int supplyPos, struct gameState *state) {
    -:  273:  int who;
  171:  274:  if (DEBUG){
    -:  275:    printf("Entering buyCard...\n");
    -:  276:  }
    -:  277:
    -:  278:  // I don't know what to do about the phase thing.
    -:  279:
  171:  280:  who = state->whoseTurn;
    -:  281:
  171:  282:  if (state->numBuys < 1){
#####:  283:    if (DEBUG)
    -:  284:      printf("You do not have any buys left\n");
```

```
#####:  285:    return -1;
  171:  286:  } else if (supplyCount(supplyPos, state) <1){
   36:  287:    if (DEBUG)
    -:  288:      printf("There are not any of that type of card left\n");
   36:  289:    return -1;
  135:  290:  } else if (state->coins < getCost(supplyPos)){
   25:  291:    if (DEBUG)
    -:  292:      printf("You do not have enough money to buy that. You have %d coins.\n", state->coins);
   25:  293:    return -1;
    -:  294:  } else {
  110:  295:    state->phase=1;
    -:  296:    //state->supplyCount[supplyPos]--;
  110:  297:    gainCard(supplyPos, state, 0, who); //card goes in discard, this might be wrong.. (2 means goes into hand, 0 goes into discard)
    -:  298:
  110:  299:    state->coins = (state->coins) - (getCost(supplyPos));
  110:  300:    state->numBuys--;
  110:  301:    if (DEBUG)
    -:  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);
    -:  303:  }
    -:  304:
    -:  305:  //state->discard[who][state->discardCount[who]] = supplyPos;
    -:  306:  //state->discardCount[who]++;
    -:  307:
  110:  308:  return 0;
    -:  309:}
    -:  310:
 2225:  311:int numHandCards(struct gameState *state) {
```

```
 2225:  312:  return state->handCount[ whoseTurn(state) ];
   -:  313:}
   -:  314:
 3608:  315:int handCard(int handPos, struct gameState *state) {
 3608:  316:  int currentPlayer = whoseTurn(state);
 3608:  317:  return state->hand[currentPlayer][handPos];
   -:  318:}
   -:  319:
  298:  320:int supplyCount(int card, struct gameState *state) {
  298:  321:  return state->supplyCount[card];
   -:  322:}
   -:  323:
#####:  324:int fullDeckCount(int player, int card, struct gameState *state) {
   -:  325:  int i;
#####:  326:  int count = 0;
   -:  327:
#####:  328:  for (i = 0; i < state->deckCount[player]; i++)
   -:  329:    {
#####:  330:      if (state->deck[player][i] == card) count++;
   -:  331:    }
   -:  332:
#####:  333:  for (i = 0; i < state->handCount[player]; i++)
   -:  334:    {
#####:  335:      if (state->hand[player][i] == card) count++;
   -:  336:    }
   -:  337:
#####:  338:  for (i = 0; i < state->discardCount[player]; i++)
   -:  339:    {
#####:  340:      if (state->discard[player][i] == card) count++;
```

```
        -:  341:    }
        -:  342:
    #####:  343:  return count;
        -:  344:}
        -:  345:
     6299:  346:int whoseTurn(struct gameState *state) {
     6299:  347:  return state->whoseTurn;
        -:  348:}
        -:  349:
      171:  350:int endTurn(struct gameState *state) {
        -:  351:  int k;
        -:  352:  int i;
      171:  353:  int currentPlayer = whoseTurn(state);
        -:  354:
        -:  355:  //Discard hand
     1051:  356:  for (i = 0; i < state->handCount[currentPlayer]; i++){
      880:  357:    state->discard[currentPlayer][state->discardCount[currentPlayer]++] = state->hand[currentPlayer][i];//Discard
      880:  358:    state->hand[currentPlayer][i] = -1;//Set card to -1
        -:  359:  }
      171:  360:  state->handCount[currentPlayer] = 0;//Reset hand count
        -:  361:
        -:  362:  //Code for determining the player
      171:  363:  if (currentPlayer < (state->numPlayers - 1)){
       86:  364:    state->whoseTurn = currentPlayer + 1;//Still safe to increment
        -:  365:  }
        -:  366:  else{
       85:  367:    state->whoseTurn = 0;//Max player has been reached, loop back around to player 1
        -:  368:  }
```

```
  -:  369:
171:  370:  state->outpostPlayed = 0;
171:  371:  state->phase = 0;
171:  372:  state->numActions = 1;
171:  373:  state->coins = 0;
171:  374:  state->numBuys = 1;
171:  375:  state->playedCardCount = 0;
171:  376:  state->handCount[state->whoseTurn] = 0;
  -:  377:
  -:  378:  //int k; move to top
  -:  379:  //Next player draws hand
1026:  380:  for (k = 0; k < 5; k++){
 855:  381:    drawCard(state->whoseTurn, state);//Draw a card
  -:  382:  }
  -:  383:
  -:  384:  //Update money
171:  385:  updateCoins(state->whoseTurn, state , 0);
  -:  386:
171:  387:  return 0;
  -:  388:}
  -:  389:
172:  390:int isGameOver(struct gameState *state) {
  -:  391:  int i;
  -:  392:  int j;
  -:  393:
  -:  394:  //if stack of Province cards is empty, the game ends
172:  395:  if (state->supplyCount[province] == 0)
  -:  396:    {
  1:  397:      return 1;
```

```
  -:  398:   }
  -:  399:
  -:  400:  //if three supply pile are at 0, the game ends
171:  401:  j = 0;
4446:  402:  for (i = 0; i < 25; i++)
  -:  403:   {
4275:  404:     if (state->supplyCount[i] == 0)
  -:  405:       {
211:  406:      j++;
  -:  407:       }
  -:  408:   }
171:  409:  if ( j >= 3)
  -:  410:   {
#####:  411:     return 1;
  -:  412:   }
  -:  413:
171:  414:  return 0;
  -:  415:}
  -:  416:
173:  417:int scoreFor (int player, struct gameState *state) {
  -:  418:
  -:  419:  int i;
173:  420:  int score = 0;
  -:  421:  //score from hand
178:  422:  for (i = 0; i < state->handCount[player]; i++)
  -:  423:   {
  5:  424:     if (state->hand[player][i] == curse) { score = score - 1; };
  5:  425:     if (state->hand[player][i] == estate) { score = score + 1; };
  5:  426:     if (state->hand[player][i] == duchy) { score = score + 3; };
```

```
      5:  427:     if (state->hand[player][i] == province) { score = score + 6; };
      5:  428:     if (state->hand[player][i] == great_hall) { score = score + 1; };
      5:  429:     if (state->hand[player][i] == gardens) { score = score + ( fullDeckCount(player, 0, state) /
10 ); };
      -:  430:   }
      -:  431:
      -:  432:  //score from discard
   3265:  433:  for (i = 0; i < state->discardCount[player]; i++)
      -:  434:   {
   3092:  435:     if (state->discard[player][i] == curse) { score = score - 1; };
   3092:  436:     if (state->discard[player][i] == estate) { score = score + 1; };
   3092:  437:     if (state->discard[player][i] == duchy) { score = score + 3; };
   3092:  438:     if (state->discard[player][i] == province) { score = score + 6; };
   3092:  439:     if (state->discard[player][i] == great_hall) { score = score + 1; };
   3092:  440:     if (state->discard[player][i] == gardens) { score = score + ( fullDeckCount(player, 0,
state) / 10 ); };
      -:  441:   }
      -:  442:
      -:  443:  //score from deck
   3265:  444:  for (i = 0; i < state->discardCount[player]; i++)
      -:  445:   {
   3092:  446:     if (state->deck[player][i] == curse) { score = score - 1; };
   3092:  447:     if (state->deck[player][i] == estate) { score = score + 1; };
   3092:  448:     if (state->deck[player][i] == duchy) { score = score + 3; };
   3092:  449:     if (state->deck[player][i] == province) { score = score + 6; };
   3092:  450:     if (state->deck[player][i] == great_hall) { score = score + 1; };
   3092:  451:     if (state->deck[player][i] == gardens) { score = score + ( fullDeckCount(player, 0, state) /
10 ); };
      -:  452:   }
      -:  453:
```

```
 173:  454:  return score;
   -:  455:}
   -:  456:
#####:  457:int getWinners(int players[MAX_PLAYERS], struct gameState *state) {
   -:  458:  int i;
   -:  459:  int j;
   -:  460:  int highScore;
   -:  461:  int currentPlayer;
   -:  462:
   -:  463:  //get score for each player
#####:  464:  for (i = 0; i < MAX_PLAYERS; i++)
   -:  465:    {
   -:  466:      //set unused player scores to -9999
#####:  467:      if (i >= state->numPlayers)
   -:  468:        {
#####:  469:      players[i] = -9999;
   -:  470:        }
   -:  471:      else
   -:  472:        {
#####:  473:      players[i] = scoreFor (i, state);
   -:  474:        }
   -:  475:    }
   -:  476:
   -:  477:  //find highest score
#####:  478:  j = 0;
#####:  479:  for (i = 0; i < MAX_PLAYERS; i++)
   -:  480:    {
#####:  481:      if (players[i] > players[j])
   -:  482:        {
```

```
#####:  483:    j = i;
   -:  484:       }
   -:  485:   }
#####:  486:  highScore = players[j];
   -:  487:
   -:  488:  //add 1 to players who had less turns
#####:  489:  currentPlayer = whoseTurn(state);
#####:  490:  for (i = 0; i < MAX_PLAYERS; i++)
   -:  491:   {
#####:  492:     if ( players[i] == highScore && i > currentPlayer )
   -:  493:       {
#####:  494:    players[i]++;
   -:  495:       }
   -:  496:   }
   -:  497:
   -:  498:  //find new highest score
#####:  499:  j = 0;
#####:  500:  for (i = 0; i < MAX_PLAYERS; i++)
   -:  501:   {
#####:  502:     if ( players[i] > players[j] )
   -:  503:       {
#####:  504:    j = i;
   -:  505:       }
   -:  506:   }
#####:  507:  highScore = players[j];
   -:  508:
   -:  509:  //set winners in array to 1 and rest to 0
#####:  510:  for (i = 0; i < MAX_PLAYERS; i++)
   -:  511:   {
```

```
#####:  512:     if ( players[i] == highScore )
   -:  513:       {
#####:  514:    players[i] = 1;
   -:  515:       }
   -:  516:     else
   -:  517:       {
#####:  518:    players[i] = 0;
   -:  519:       }
   -:  520:   }
   -:  521:
#####:  522:  return 0;
   -:  523:}
   -:  524:
  905:  525:int drawCard(int player, struct gameState *state)
   -:  526:{      int count;
   -:  527:  int deckCounter;
  905:  528:  if (state->deckCount[player] <= 0){//Deck is empty
   -:  529:
   -:  530:    //Step 1 Shuffle the discard pile back into a deck
   -:  531:    int i;
   -:  532:    //Move discard to deck
  960:  533:    for (i = 0; i < state->discardCount[player];i++){
  919:  534:      state->deck[player][i] = state->discard[player][i];
  919:  535:      state->discard[player][i] = -1;
   -:  536:    }
   -:  537:
   41:  538:    state->deckCount[player] = state->discardCount[player];
   41:  539:    state->discardCount[player] = 0;//Reset discard
   -:  540:
```

```
  -:  541:    //Shufffle the deck
 41:  542:    shuffle(player, state);//Shuffle the deck up and make it so that we can draw
  -:  543:
 41:  544:    if (DEBUG){//Debug statements
  -:  545:      printf("Deck count now: %d\n", state->deckCount[player]);
  -:  546:    }
  -:  547:
 41:  548:    state->discardCount[player] = 0;
  -:  549:
  -:  550:    //Step 2 Draw Card
 41:  551:    count = state->handCount[player];//Get current player's hand count
  -:  552:
 41:  553:    if (DEBUG){//Debug statements
  -:  554:      printf("Current hand count: %d\n", count);
  -:  555:    }
  -:  556:
 41:  557:    deckCounter = state->deckCount[player];//Create a holder for the deck count
  -:  558:
 41:  559:    if (deckCounter == 0)
#####:  560:      return -1;
  -:  561:
 41:  562:    state->hand[player][count] = state->deck[player][deckCounter - 1];//Add card to hand
 41:  563:    state->deckCount[player]--;
 41:  564:    state->handCount[player]++;//Increment hand count
  -:  565:  }
  -:  566:
  -:  567:  else{
864:  568:    int count = state->handCount[player];//Get current hand count for player
  -:  569:    int deckCounter;
```

```
    864:  570:    if (DEBUG){//Debug statements
     -:  571:      printf("Current hand count: %d\n", count);
     -:  572:    }
     -:  573:
    864:  574:    deckCounter = state->deckCount[player];//Create holder for the deck count
    864:  575:    state->hand[player][count] = state->deck[player][deckCounter - 1];//Add card to the
hand
    864:  576:    state->deckCount[player]--;
    864:  577:    state->handCount[player]++;//Increment hand count
     -:  578:  }
     -:  579:
    905:  580:  return 0;
     -:  581:}
     -:  582:
    253:  583:int getCost(int cardNumber)
     -:  584:{
    253:  585:  switch( cardNumber )
     -:  586:    {
     -:  587:    case curse:
  #####:  588:      return 0;
     -:  589:    case estate:
     18:  590:      return 2;
     -:  591:    case duchy:
     17:  592:      return 5;
     -:  593:    case province:
     16:  594:      return 8;
     -:  595:    case copper:
     29:  596:      return 0;
     -:  597:    case silver:
```

```
  60:  598:     return 3;
   -:  599:   case gold:
  19:  600:     return 6;
   -:  601:   case adventurer:
   7:  602:     return 6;
   -:  603:   case council_room:
  11:  604:     return 5;
   -:  605:   case feast:
#####:  606:     return 4;
   -:  607:   case gardens:
#####:  608:     return 4;
   -:  609:   case mine:
  13:  610:     return 5;
   -:  611:   case remodel:
   9:  612:     return 4;
   -:  613:   case smithy:
#####:  614:     return 4;
   -:  615:   case village:
#####:  616:     return 3;
   -:  617:   case baron:
  13:  618:     return 4;
   -:  619:   case great_hall:
#####:  620:     return 3;
   -:  621:   case minion:
#####:  622:     return 5;
   -:  623:   case steward:
#####:  624:     return 3;
   -:  625:   case tribute:
   7:  626:     return 5;
```

```
        -:  627:    case ambassador:
       10:  628:      return 3;
        -:  629:    case cutpurse:
    #####:  630:      return 4;
        -:  631:    case embargo:
    #####:  632:      return 2;
        -:  633:    case outpost:
        6:  634:      return 5;
        -:  635:    case salvager:
    #####:  636:      return 4;
        -:  637:    case sea_hag:
       11:  638:      return 4;
        -:  639:    case treasure_map:
        7:  640:      return 4;
        -:  641:    }
        -:  642:
    #####:  643:  return -1;
        -:  644:}
        -:  645:
       87:  646:int cardEffect(int card, int choice1, int choice2, int choice3, struct gameState *state, int handPos, int *bonus)
        -:  647:{
        -:  648:  int i;
        -:  649:  int j;
        -:  650:  int k;
        -:  651:  int x;
        -:  652:  int index;
       87:  653:  int currentPlayer = whoseTurn(state);
       87:  654:  int nextPlayer = currentPlayer + 1;
```

```
  -:  655:
 87:  656:  int tributeRevealedCards[2] = {-1, -1};
  -:  657:  int temphand[MAX_HAND];// moved above the if statement
 87:  658:  int drawntreasure=0;
  -:  659:  int cardDrawn;
 87:  660:  int z = 0;// this is the counter for the temp hand
 87:  661:  if (nextPlayer > (state->numPlayers - 1)){
 49:  662:    nextPlayer = 0;
  -:  663:  }
  -:  664:
  -:  665:
  -:  666:  //uses switch to select card and perform actions
 87:  667:  switch( card )
  -:  668:    {
  -:  669:    case adventurer:
  7:  670:                return cardAdventurer(state);
  -:  671:
  -:  672:    case council_room:
  2:  673:                return cardCouncilRoom(state, handPos);
  -:  674:
  -:  675:    case feast:
#####:  676:                return cardFeast(choice1, state);
  -:  677:
  -:  678:    case gardens:
#####:  679:                return cardGardens();
  -:  680:
  -:  681:    case mine:
 24:  682:                return cardMine(choice1, choice2, state, handPos);
  -:  683:
```

```
   -:  684:    case remodel:
   4:  685:                    return cardRemodel(choice1, choice2, state, handPos);
   -:  686:
   -:  687:    case smithy:
   -:  688:      //+3 Cards
#####:  689:      for (i = 0; i < 3; i++)
   -:  690:        {
#####:  691:      drawCard(currentPlayer, state);
   -:  692:        }
   -:  693:
   -:  694:      //discard card from hand
#####:  695:      discardCard(handPos, currentPlayer, state, 0);
#####:  696:      return 0;
   -:  697:
   -:  698:    case village:
   -:  699:      //+1 Card
#####:  700:      drawCard(currentPlayer, state);
   -:  701:
   -:  702:      //+2 Actions
#####:  703:      state->numActions = state->numActions + 2;
   -:  704:
   -:  705:      //discard played card from hand
#####:  706:      discardCard(handPos, currentPlayer, state, 0);
#####:  707:      return 0;
   -:  708:
   -:  709:    case baron:
  12:  710:      state->numBuys++;//Increase buys by 1!
  12:  711:      if (choice1 > 0){//Boolean true or going to discard an estate
   1:  712:        int p = 0;//Iterator for hand!
```

```
    1:  713:       int card_not_discarded = 1;//Flag for discard set!
    4:  714:       while(card_not_discarded){
    2:  715:         if (state->hand[currentPlayer][p] == estate){//Found an estate card!
    1:  716:           state->coins += 4;//Add 4 coins to the amount of coins
    1:  717:           state->discard[currentPlayer][state->discardCount[currentPlayer]] = state-
>hand[currentPlayer][p];
    1:  718:           state->discardCount[currentPlayer]++;
    5:  719:           for (;p < state->handCount[currentPlayer]; p++){
    4:  720:             state->hand[currentPlayer][p] = state->hand[currentPlayer][p+1];
    -:  721:           }
    1:  722:           state->hand[currentPlayer][state->handCount[currentPlayer]] = -1;
    1:  723:           state->handCount[currentPlayer]--;
    1:  724:           card_not_discarded = 0;//Exit the loop
    -:  725:         }
    1:  726:         else if (p > state->handCount[currentPlayer]){
#####:  727:           if(DEBUG) {
    -:  728:             printf("No estate cards in your hand, invalid choice\n");
    -:  729:             printf("Must gain an estate if there are any\n");
    -:  730:           }
#####:  731:           if (supplyCount(estate, state) > 0){
#####:  732:             gainCard(estate, state, 0, currentPlayer);
#####:  733:             state->supplyCount[estate]--;//Decrement estates
#####:  734:             if (supplyCount(estate, state) == 0){
#####:  735:                   isGameOver(state);
    -:  736:             }
    -:  737:           }
#####:  738:           card_not_discarded = 0;//Exit the loop
    -:  739:         }
    -:  740:
```

```
   -:  741:         else{
   1:  742:           p++;//Next card
   -:  743:         }
   -:  744:       }
   -:  745:     }
   -:  746:
   -:  747:     else{
  11:  748:       if (supplyCount(estate, state) > 0){
#####:  749:     gainCard(estate, state, 0, currentPlayer);//Gain an estate
#####:  750:     state->supplyCount[estate]--;//Decrement Estates
#####:  751:     if (supplyCount(estate, state) == 0){
#####:  752:       isGameOver(state);
   -:  753:         }
   -:  754:       }
   -:  755:     }
   -:  756:
   -:  757:
  12:  758:     return 0;
   -:  759:
   -:  760:   case great_hall:
   -:  761:     //+1 Card
#####:  762:     drawCard(currentPlayer, state);
   -:  763:
   -:  764:     //+1 Actions
#####:  765:     state->numActions++;
   -:  766:
   -:  767:     //discard card from hand
#####:  768:     discardCard(handPos, currentPlayer, state, 0);
#####:  769:     return 0;
```

```
    -:  770:
    -:  771:    case minion:
    -:  772:      //+1 action
#####:  773:      state->numActions++;
    -:  774:
    -:  775:      //discard card from hand
#####:  776:      discardCard(handPos, currentPlayer, state, 0);
    -:  777:
#####:  778:      if (choice1)                //+2 coins
    -:  779:        {
#####:  780:    state->coins = state->coins + 2;
    -:  781:        }
    -:  782:
#####:  783:      else if (choice2)                     //discard hand, redraw 4, other players with 5+ cards discard hand and draw 4
    -:  784:        {
    -:  785:          //discard hand
#####:  786:    while(numHandCards(state) > 0)
    -:  787:          {
#####:  788:        discardCard(handPos, currentPlayer, state, 0);
    -:  789:          }
    -:  790:
    -:  791:          //draw 4
#####:  792:    for (i = 0; i < 4; i++)
    -:  793:          {
#####:  794:        drawCard(currentPlayer, state);
    -:  795:          }
    -:  796:
    -:  797:          //other players discard hand and redraw if hand size > 4
```

```
#####:  798:    for (i = 0; i < state->numPlayers; i++)
   -:  799:          {
#####:  800:        if (i != currentPlayer)
   -:  801:                  {
#####:  802:                if ( state->handCount[i] > 4 )
   -:  803:                  {
   -:  804:                    //discard hand
#####:  805:                    while( state->handCount[i] > 0 )
   -:  806:                          {
#####:  807:                            discardCard(handPos, i, state, 0);
   -:  808:                          }
   -:  809:
   -:  810:                    //draw 4
#####:  811:                    for (j = 0; j < 4; j++)
   -:  812:                          {
#####:  813:                            drawCard(i, state);
   -:  814:                          }
   -:  815:                  }
   -:  816:                }
   -:  817:          }
   -:  818:
   -:  819:       }
#####:  820:     return 0;
   -:  821:
   -:  822:    case steward:
#####:  823:     if (choice1 == 1)
   -:  824:       {
   -:  825:         //+2 cards
#####:  826:     drawCard(currentPlayer, state);
```

```
#####:  827:    drawCard(currentPlayer, state);
   -:  828:      }
#####:  829:    else if (choice1 == 2)
   -:  830:      {
   -:  831:        //+2 coins
#####:  832:    state->coins = state->coins + 2;
   -:  833:      }
   -:  834:    else
   -:  835:      {
   -:  836:        //trash 2 cards in hand
#####:  837:    discardCard(choice2, currentPlayer, state, 1);
#####:  838:    discardCard(choice3, currentPlayer, state, 1);
   -:  839:      }
   -:  840:
   -:  841:    //discard card from hand
#####:  842:    discardCard(handPos, currentPlayer, state, 0);
#####:  843:    return 0;
   -:  844:
   -:  845:  case tribute:
    9:  846:    if ((state->discardCount[nextPlayer] + state->deckCount[nextPlayer]) <= 1){
#####:  847:  if (state->deckCount[nextPlayer] > 0){
#####:  848:    tributeRevealedCards[0] = state->deck[nextPlayer][state->deckCount[nextPlayer]-1];
#####:  849:    state->deckCount[nextPlayer]--;
   -:  850:      }
#####:  851:  else if (state->discardCount[nextPlayer] > 0){
#####:  852:    tributeRevealedCards[0] = state->discard[nextPlayer][state->discardCount[nextPlayer]-
1];
#####:  853:    state->discardCount[nextPlayer]--;
   -:  854:      }
```

```
        -:  855:       else{
        -:  856:         //No Card to Reveal
    #####:  857:    if (DEBUG){
        -:  858:          printf("No cards to reveal\n");
        -:  859:        }
        -:  860:      }
        -:  861:    }
        -:  862:
        -:  863:    else{
        9:  864:      if (state->deckCount[nextPlayer] == 0){
       16:  865:       for (i = 0; i < state->discardCount[nextPlayer]; i++){
       15:  866:         state->deck[nextPlayer][i] = state->discard[nextPlayer][i];//Move to deck
       15:  867:         state->deckCount[nextPlayer]++;
       15:  868:         state->discard[nextPlayer][i] = -1;
       15:  869:         state->discardCount[nextPlayer]--;
        -:  870:       }
        -:  871:
        1:  872:       shuffle(nextPlayer,state);//Shuffle the deck
        -:  873:      }
        9:  874:      tributeRevealedCards[0] = state->deck[nextPlayer][state->deckCount[nextPlayer]-1];
        9:  875:      state->deck[nextPlayer][state->deckCount[nextPlayer]--] = -1;
        9:  876:      state->deckCount[nextPlayer]--;
        9:  877:      tributeRevealedCards[1] = state->deck[nextPlayer][state->deckCount[nextPlayer]-1];
        9:  878:      state->deck[nextPlayer][state->deckCount[nextPlayer]--] = -1;
        9:  879:      state->deckCount[nextPlayer]--;
        -:  880:    }
        -:  881:
        9:  882:    if (tributeRevealedCards[0] == tributeRevealedCards[1]){//If we have a duplicate card,
just drop one
```

```
        2:  883:       state->playedCards[state->playedCardCount] = tributeRevealedCards[1];
        2:  884:       state->playedCardCount++;
        2:  885:       tributeRevealedCards[1] = -1;
       -:  886:     }
       -:  887:
       36:  888:     for (i = 0; i <= 2; i ++){
       37:  889:     if (tributeRevealedCards[i] == copper || tributeRevealedCards[i] == silver ||
tributeRevealedCards[i] == gold){//Treasure cards
       10:  890:       state->coins += 2;
       -:  891:       }
       -:  892:
       24:  893:     else if (tributeRevealedCards[i] == estate || tributeRevealedCards[i] == duchy ||
tributeRevealedCards[i] == province || tributeRevealedCards[i] == gardens || tributeRevealedCards[i] ==
great_hall){//Victory Card Found
        7:  894:       drawCard(currentPlayer, state);
        7:  895:       drawCard(currentPlayer, state);
       -:  896:       }
       -:  897:       else{//Action Card
       10:  898:       state->numActions = state->numActions + 2;
       -:  899:       }
       -:  900:     }
       -:  901:
        9:  902:     return 0;
       -:  903:
       -:  904:    case ambassador:
       15:  905:     j = 0;                  //used to check if player has enough cards to discard
       -:  906:
       15:  907:     if (choice2 > 2 || choice2 < 0)
       -:  908:       {
       15:  909:       return -1;
```

```
    -:  910:       }
    -:  911:
#####:  912:     if (choice1 == handPos)
    -:  913:       {
#####:  914:     return -1;
    -:  915:       }
    -:  916:
#####:  917:     for (i = 0; i < state->handCount[currentPlayer]; i++)
    -:  918:       {
#####:  919:     if (i != handPos && i == state->hand[currentPlayer][choice1] && i != choice1)
    -:  920:         {
#####:  921:       j++;
    -:  922:         }
    -:  923:       }
#####:  924:     if (j < choice2)
    -:  925:       {
#####:  926:     return -1;
    -:  927:       }
    -:  928:
#####:  929:     if (DEBUG)
    -:  930:       printf("Player %d reveals card number: %d\n", currentPlayer, state->hand[currentPlayer][choice1]);
    -:  931:
    -:  932:     //increase supply count for choosen card by amount being discarded
#####:  933:     state->supplyCount[state->hand[currentPlayer][choice1]] += choice2;
    -:  934:
    -:  935:     //each other player gains a copy of revealed card
#####:  936:     for (i = 0; i < state->numPlayers; i++)
    -:  937:       {
```

```
#####:  938:    if (i != currentPlayer)
   -:  939:          {
#####:  940:        gainCard(state->hand[currentPlayer][choice1], state, 0, i);
   -:  941:          }
   -:  942:       }
   -:  943:
   -:  944:     //discard played card from hand
#####:  945:     discardCard(handPos, currentPlayer, state, 0);
   -:  946:
   -:  947:     //trash copies of cards returned to supply
#####:  948:     for (j = 0; j < choice2; j++)
   -:  949:       {
#####:  950:     for (i = 0; i < state->handCount[currentPlayer]; i++)
   -:  951:          {
#####:  952:        if (state->hand[currentPlayer][i] == state->hand[currentPlayer][choice1])
   -:  953:                {
#####:  954:                  discardCard(i, currentPlayer, state, 1);
#####:  955:                  break;
   -:  956:                }
   -:  957:          }
   -:  958:       }
   -:  959:
#####:  960:     return 0;
   -:  961:
   -:  962:    case cutpurse:
   -:  963:
#####:  964:     updateCoins(currentPlayer, state, 2);
#####:  965:     for (i = 0; i < state->numPlayers; i++)
   -:  966:       {
```

```
#####:  967:    if (i != currentPlayer)
   -:  968:          {
#####:  969:       for (j = 0; j < state->handCount[i]; j++)
   -:  970:                  {
#####:  971:                    if (state->hand[i][j] == copper)
   -:  972:                      {
#####:  973:                        discardCard(j, i, state, 0);
#####:  974:                        break;
   -:  975:                      }
#####:  976:                    if (j == state->handCount[i])
   -:  977:                      {
#####:  978:                        for (k = 0; k < state->handCount[i]; k++)
   -:  979:                            {
#####:  980:                              if (DEBUG)
   -:  981:                                printf("Player %d reveals card number %d\n", i, state->hand[i][k]);
   -:  982:                            }
#####:  983:                        break;
   -:  984:                      }
   -:  985:                  }
   -:  986:
   -:  987:          }
   -:  988:
   -:  989:       }
   -:  990:
   -:  991:     //discard played card from hand
#####:  992:     discardCard(handPos, currentPlayer, state, 0);
   -:  993:
#####:  994:     return 0;
   -:  995:
```

```
   -:  996:
   -:  997:    case embargo:
   -:  998:      //+2 Coins
#####:  999:      state->coins = state->coins + 2;
   -: 1000:
   -: 1001:      //see if selected pile is in play
#####: 1002:      if ( state->supplyCount[choice1] == -1 )
   -: 1003:      {
#####: 1004:    return -1;
   -: 1005:      }
   -: 1006:
   -: 1007:      //add embargo token to selected supply pile
#####: 1008:      state->embargoTokens[choice1]++;
   -: 1009:
   -: 1010:      //trash card
#####: 1011:      discardCard(handPos, currentPlayer, state, 1);
#####: 1012:      return 0;
   -: 1013:
   -: 1014:    case outpost:
   -: 1015:      //set outpost flag
   2: 1016:      state->outpostPlayed++;
   -: 1017:
   -: 1018:      //discard card
   2: 1019:      discardCard(handPos, currentPlayer, state, 0);
   2: 1020:      return 0;
   -: 1021:
   -: 1022:    case salvager:
   -: 1023:      //+1 buy
#####: 1024:      state->numBuys++;
```

```
   -: 1025:
#####: 1026:     if (choice1)
   -: 1027:     {
   -: 1028:       //gain coins equal to trashed card
#####: 1029:   state->coins = state->coins + getCost( handCard(choice1, state) );
   -: 1030:       //trash card
#####: 1031:   discardCard(choice1, currentPlayer, state, 1);
   -: 1032:     }
   -: 1033:
   -: 1034:     //discard card
#####: 1035:     discardCard(handPos, currentPlayer, state, 0);
#####: 1036:     return 0;
   -: 1037:
   -: 1038:   case sea_hag:
  12: 1039:     for (i = 0; i < state->numPlayers; i++){
   8: 1040:     if (i != currentPlayer){
   4: 1041:       state->discard[i][state->discardCount[i]] = state->deck[i][state->deckCount[i]--];
                  state->deckCount[i]--;
   4: 1042:       state->discardCount[i]++;
   4: 1043:       state->deck[i][state->deckCount[i]--] = curse;//Top card now a curse
   -: 1044:     }
   -: 1045:   }
   4: 1046:     return 0;
   -: 1047:
   -: 1048:   case treasure_map:
   -: 1049:     //search hand for another treasure_map
   8: 1050:     index = -1;
  44: 1051:     for (i = 0; i < state->handCount[currentPlayer]; i++)
   -: 1052:     {
```

```
  37: 1053:       if (state->hand[currentPlayer][i] == treasure_map && i != handPos)
   -: 1054:         {
   1: 1055:           index = i;
   1: 1056:           break;
   -: 1057:         }
   -: 1058:       }
   8: 1059:     if (index > -1)
   -: 1060:     {
   -: 1061:       //trash both treasure cards
   1: 1062:       discardCard(handPos, currentPlayer, state, 1);
   1: 1063:       discardCard(index, currentPlayer, state, 1);
   -: 1064:
   -: 1065:       //gain 4 Gold cards
   5: 1066:       for (i = 0; i < 4; i++)
   -: 1067:         {
   4: 1068:           gainCard(gold, state, 1, currentPlayer);
   -: 1069:         }
   -: 1070:
   -: 1071:       //return success
   1: 1072:       return 1;
   -: 1073:     }
   -: 1074:
   -: 1075:     //no second treasure_map found in hand
   7: 1076:     return -1;
   -: 1077:   }
   -: 1078:
#####: 1079:  return -1;
   -: 1080:}
   -: 1081:
```

```
   7: 1082:int cardAdventurer(struct gameState *state)
   -: 1083:{
   7: 1084:     int currentPlayer = whoseTurn(state);
   -: 1085:     int temphand[MAX_HAND];// moved above the if statement
   7: 1086:     int drawntreasure = 0;
   -: 1087:     int cardDrawn;
   7: 1088:     int z = 0;
   -: 1089:
  35: 1090:     while(drawntreasure < 2)
   -: 1091:     {
  21: 1092:          if (state->deckCount[currentPlayer] < 1)
   -: 1093:          {//if the deck is empty we need to shuffle discard and add to deck
#####: 1094:               shuffle(currentPlayer, state);
   -: 1095:          }
   -: 1096:
   -: 1097:          //top card of hand is most recently drawn card.
  21: 1098:          drawCard(currentPlayer, state);
  21: 1099:          cardDrawn = state->hand[currentPlayer][state->handCount[currentPlayer] - 1];
   -: 1100:
  35: 1101:          if (cardDrawn == copper || cardDrawn == silver || cardDrawn == gold)
   -: 1102:          {
  14: 1103:               drawntreasure++;
   -: 1104:          }
   -: 1105:          else
   -: 1106:          {
   7: 1107:               temphand[z] = cardDrawn;
   -: 1108:               //this should just remove the top card (the most recently drawn one).
   7: 1109:               state->handCount[currentPlayer]--;
   7: 1110:               z++;
```

```
        -: 1111:                }
        -: 1112:        }
       21: 1113:    while((z - 1) >= 0){
        -: 1114:                // discard all cards in play that have been drawn
        7: 1115:                state->discard[currentPlayer][state->discardCount[currentPlayer]++] =
temphand[z - 1];
        7: 1116:                z = z - 1;
        -: 1117:    }
        7: 1118:    return 0;
        -: 1119:}
        -: 1120:
        2: 1121:int cardCouncilRoom(struct gameState *state, int handPos)
        -: 1122:{
        -: 1123:       int i;
        2: 1124:       int currentPlayer = whoseTurn(state);
        -: 1125:
        -: 1126:       //+4 Cards
       10: 1127:       for (i = 0; i < 4; i++)
        -: 1128:       {
        8: 1129:                drawCard(currentPlayer, state);
        -: 1130:       }
        -: 1131:
        -: 1132:       //+1 Buy
        2: 1133:       state->numBuys++;
        -: 1134:
        -: 1135:       //Each other player draws a card
        6: 1136:       for (i = 0; i < state->numPlayers; i++)
        -: 1137:       {
        4: 1138:                if ( i != currentPlayer )
```

```
   -: 1139:                {
   2: 1140:                        drawCard(i, state);
   -: 1141:                }
   -: 1142:        }
   -: 1143:
   -: 1144:        //put played card in played card pile
   2: 1145:        discardCard(handPos, currentPlayer, state, 0);
   -: 1146:
   2: 1147:        return 0;
   -: 1148:}
   -: 1149:
#####: 1150:int cardFeast(int choice1, struct gameState *state)
   -: 1151:{
   -: 1152:        int i;
   -: 1153:        int x;
#####: 1154:    int currentPlayer = whoseTurn(state);
   -: 1155:        int temphand[MAX_HAND];
   -: 1156:
   -: 1157:        //gain card with cost up to 5
   -: 1158:        //Backup hand
#####: 1159:    for (i = 0; i <= state->handCount[currentPlayer]; i++){
#####: 1160:                temphand[i] = state->hand[currentPlayer][i];//Backup card
#####: 1161:                state->hand[currentPlayer][i] = -1;//Set to nothing
   -: 1162:        }
   -: 1163:        //Backup hand
   -: 1164:
   -: 1165:        //Update Coins for Buy
#####: 1166:    updateCoins(currentPlayer, state, 5);
   -: 1167:        //x = 1;//Condition to loop on
```

```
    #####: 1168:  x = -1;
       -: 1169:
    #####: 1170:  while(x == 1)
       -: 1171:      {//Buy one card
    #####: 1172:              if (supplyCount(choice1, state) <= 0)
       -: 1173:              {
    #####: 1174:                      if (DEBUG)
       -: 1175:                      {
       -: 1176:                              printf("None of that card left, sorry!\n");
       -: 1177:                      }
    #####: 1178:                      if (DEBUG){
       -: 1179:                              printf("Cards Left: %d\n", supplyCount(choice1, state));
       -: 1180:                      }
       -: 1181:              }
    #####: 1182:              else if (state->coins < getCost(choice1))
       -: 1183:              {
    #####: 1184:                      printf("That card is too expensive!\n");
       -: 1185:
    #####: 1186:                      if (DEBUG)
       -: 1187:                      {
       -: 1188:                              printf("Coins: %d < %d\n", state->coins, getCost(choice1));
       -: 1189:                      }
       -: 1190:              }
       -: 1191:              else
       -: 1192:              {
    #####: 1193:                      if (DEBUG){
       -: 1194:                              printf("Deck Count: %d\n", state->handCount[currentPlayer] +
state->deckCount[currentPlayer] + state->discardCount[currentPlayer]);
       -: 1195:                      }
```

```
    -: 1196:
#####: 1197:                    gainCard(choice1, state, 0, currentPlayer);//Gain the card
#####: 1198:                    x = 0;//No more buying cards
    -: 1199:
#####: 1200:                    if (DEBUG)
    -: 1201:                    {
    -: 1202:                            printf("Deck Count: %d\n", state->handCount[currentPlayer] +
state->deckCount[currentPlayer] + state->discardCount[currentPlayer]);
    -: 1203:                    }
    -: 1204:            }
    -: 1205:    }
    -: 1206:
    -: 1207:    //Reset Hand
#####: 1208:  for (i = 0; i <= state->handCount[currentPlayer]; i++){
#####: 1209:          state->hand[currentPlayer][i] = temphand[i];
#####: 1210:          temphand[i] = -1;
    -: 1211:    }
    -: 1212:    //Reset Hand
#####: 1213:  return 0;
    -: 1214:}
    -: 1215:
#####: 1216:int cardGardens()
    -: 1217:{
    -: 1218:    //return -1;
#####: 1219:  return 1;
    -: 1220:}
    -: 1221:
   24: 1222:int cardMine(int choice1, int choice2, struct gameState *state, int handPos)
    -: 1223:{
```

```
        -: 1224:    int i;
        -: 1225:    int j;
        -: 1226:    //int currentPlayer = whoseTurn(state);
       24: 1227:    int currentPlayer = whoseTurn(state) + 1;
        -: 1228:
       24: 1229:    j = state->hand[currentPlayer][choice1];  //store card we will trash
        -: 1230:
       24: 1231:    if (state->hand[currentPlayer][choice1] < copper || state->hand[currentPlayer][choice1]
> gold)
        -: 1232:    {
       24: 1233:        return -1;
        -: 1234:    }
        -: 1235:
    #####: 1236:    if (choice2 > treasure_map || choice2 < curse)
        -: 1237:    {
    #####: 1238:        return -1;
        -: 1239:    }
        -: 1240:
    #####: 1241:    if ((getCost(state->hand[currentPlayer][choice1]) + 3) > getCost(choice2))
        -: 1242:    {
    #####: 1243:        return -1;
        -: 1244:    }
        -: 1245:
    #####: 1246:    gainCard(choice2, state, 2, currentPlayer);
        -: 1247:
        -: 1248:    //discard card from hand
    #####: 1249:    discardCard(handPos, currentPlayer, state, 0);
        -: 1250:
        -: 1251:    //discard trashed card
```

```
#####: 1252:  for (i = 0; i < state->handCount[currentPlayer]; i++)
   -: 1253:      {
#####: 1254:              if (state->hand[currentPlayer][i] == j)
   -: 1255:              {
#####: 1256:                      discardCard(i, currentPlayer, state, 0);
#####: 1257:                      break;
   -: 1258:          }
   -: 1259:      }
   -: 1260:
#####: 1261:  return 0;
   -: 1262:}
   -: 1263:
   4: 1264:int cardRemodel(int choice1, int choice2, struct gameState *state, int handPos)
   -: 1265:{
   -: 1266:      int i;
   -: 1267:      int j;
   4: 1268:      int currentPlayer = whoseTurn(state);
   -: 1269:
   4: 1270:      j = state->hand[currentPlayer][choice1];  //store card we will trash
   -: 1271:
   4: 1272:      if ((getCost(state->hand[currentPlayer][choice1]) + 2) > getCost(choice2))
   -: 1273:      {
   2: 1274:              return -1;
   -: 1275:      }
   -: 1276:
   2: 1277:      gainCard(choice2, state, 0, currentPlayer);
   -: 1278:
   -: 1279:      //discard card from hand
   2: 1280:      discardCard(handPos, currentPlayer, state, 0);
```

```
-: 1281:
-: 1282:      //discard trashed card
5: 1283:      for (i = 0; i < state->handCount[currentPlayer]; i++)
-: 1284:      {
5: 1285:              if (state->hand[currentPlayer][i] == j)
-: 1286:              {
2: 1287:                      discardCard(i, currentPlayer, state, 0);
2: 1288:                      break;
-: 1289:              }
-: 1290:      }
2: 1291:      return 0;
-: 1292:}
-: 1293:
10: 1294:int discardCard(int handPos, int currentPlayer, struct gameState *state, int trashFlag)
-: 1295:{
-: 1296:
-: 1297:  //if card is not trashed, added to Played pile
10: 1298:  if (trashFlag < 1)
-: 1299:    {
-: 1300:     //add card to played pile
8: 1301:     state->playedCards[state->playedCardCount] = state->hand[currentPlayer][handPos];
8: 1302:     state->playedCardCount++;
-: 1303:    }
-: 1304:
-: 1305:  //set played card to -1
10: 1306:  state->hand[currentPlayer][handPos] = -1;
-: 1307:
-: 1308:  //remove card from player's hand
```

```
    10: 1309:  if ( handPos == (state->handCount[currentPlayer] - 1) )       //last card in hand array is played
    -: 1310:    {
    -: 1311:      //reduce number of cards in hand
    1: 1312:      state->handCount[currentPlayer]--;
    -: 1313:    }
    9: 1314:  else if ( state->handCount[currentPlayer] == 1 ) //only one card in hand
    -: 1315:    {
    -: 1316:      //reduce number of cards in hand
#####: 1317:      state->handCount[currentPlayer]--;
    -: 1318:    }
    -: 1319:  else
    -: 1320:    {
    -: 1321:      //replace discarded card with last card in hand
    9: 1322:      state->hand[currentPlayer][handPos] = state->hand[currentPlayer][ (state->handCount[currentPlayer] - 1)];
    -: 1323:      //set last card to -1
    9: 1324:      state->hand[currentPlayer][state->handCount[currentPlayer] - 1] = -1;
    -: 1325:      //reduce number of cards in hand
    9: 1326:      state->handCount[currentPlayer]--;
    -: 1327:    }
    -: 1328:
   10: 1329:  return 0;
    -: 1330:}
    -: 1331:
  116: 1332:int gainCard(int supplyPos, struct gameState *state, int toFlag, int player)
    -: 1333:{
    -: 1334:  //Note: supplyPos is enum of choosen card
    -: 1335:
```

```
   -: 1336:  //check if supply pile is empty (0) or card is not used in game (-1)
 116: 1337:  if ( supplyCount(supplyPos, state) < 1 )
   -: 1338:    {
#####: 1339:      return -1;
   -: 1340:    }
   -: 1341:
   -: 1342:  //added card for [whoseTurn] current player:
   -: 1343:  // toFlag = 0 : add to discard
   -: 1344:  // toFlag = 1 : add to deck
   -: 1345:  // toFlag = 2 : add to hand
   -: 1346:
 116: 1347:  if (toFlag == 1)
   -: 1348:    {
   4: 1349:      state->deck[ player ][ state->deckCount[player] ] = supplyPos;
   4: 1350:      state->deckCount[player]++;
   -: 1351:    }
 112: 1352:  else if (toFlag == 2)
   -: 1353:    {
#####: 1354:      state->hand[ player ][ state->handCount[player] ] = supplyPos;
#####: 1355:      state->handCount[player]++;
   -: 1356:    }
   -: 1357:  else
   -: 1358:    {
 112: 1359:      state->discard[player][ state->discardCount[player] ] = supplyPos;
 112: 1360:      state->discardCount[player]++;
   -: 1361:    }
   -: 1362:
   -: 1363:  //decrease number in supply pile
 116: 1364:  state->supplyCount[supplyPos]--;
```

```
  -: 1365:
116: 1366:  return 0;
  -: 1367:}
  -: 1368:
211: 1369:int updateCoins(int player, struct gameState *state, int bonus)
  -: 1370:{
  -: 1371:  int i;
  -: 1372:
  -: 1373:  //reset coin count
211: 1374:  state->coins = 0;
  -: 1375:
  -: 1376:  //add coins for each Treasure card in player's hand
1291: 1377:  for (i = 0; i < state->handCount[player]; i++)
  -: 1378:    {
1080: 1379:      if (state->hand[player][i] == copper)
  -: 1380:      {
302: 1381:      state->coins += 1;
  -: 1382:      }
778: 1383:      else if (state->hand[player][i] == silver)
  -: 1384:      {
207: 1385:      state->coins += 2;
  -: 1386:      }
571: 1387:      else if (state->hand[player][i] == gold)
  -: 1388:      {
 72: 1389:      state->coins += 3;
  -: 1390:      }
  -: 1391:    }
  -: 1392:
  -: 1393:  //add bonus
```

```
211: 1394:  state->coins += bonus;
 -: 1395:
211: 1396:  return 0;
 -: 1397:}
 -: 1398:
 -: 1399:
 -: 1400://end of dominion.c
 -: 1401:
```