Assignment 2

Refactor

To extract the code from the switch statement into their own individual functions, I began by determining which variables would need to be passed to the functions. In the switch statement, many variables are declared at the beginning of the statement and do not need to be passed to the switch statement. However, to be able to use a function, each element that the function uses must be passed to the function. Therefore, each argument used within the functions must be passed to it from the action function. I cleaned up the code by running through the arguments passed and ensuring that each function has the required arguments in both the .c file and the .h file. I did not create any global variables because I know how dangerous that can be. The major changes are the introduction of the new functions: `int ambassadorPlayed(int card, int choice1, int choice2, int choice3, struct gameState *state, int handPos, int *bonus, int j, int currentPlayer) {`
And commenting out the old functions and calling the new ones within the switch statement:

```
case ambassador:
            ambassadorPlayed(card, choice1, choice2, choice3, state, handPos, bonus, j, currentPlayer);
// j = 0;  //used to check if player has enough cards to discard
```

Bugs

- Mine bug 1:
  `if (state->hand[currentPlayer][choice1] < silver || state->hand[currentPlayer][choice1] > gold) // bug 1`
  This bug introduces the issue of comparing the player choice to silver instead of copper. This would be revealed if the player chose copper, the effect would be as if the player chose something less than copper.
- Mine bug 2:
  `gainCard(choice2, state, 5, currentPlayer); // bug 2`
  This bug gives a player too many cards when the mine card is played. This would be revealed when playing the mine card after choosing a card.
- Baron bug 1:
  `for (; p < state->handCount[currentPlayer]; p++) { state->hand[currentPlayer][p] = state->hand[currentPlayer][p+2]; //bug 1 }`
  this bug sets the card at p to equal the card two in front of p as opposed to one in front of p. This could potentially lead to bugs during gameplay and out of bounds exceptions when using the Baron card.
- Baron bug 2:
  `if (supplyCount(estate, state) == 1) { //bug 2 isGameOver(state); }`
  This bug will be revealed when a player's estates are decremented after discarding an estate. This could affect the game by triggering a false end of game before either player has won or missing a true end of game when someone has won.
- Minion bug 1: `while(numHandCards(state) < 0)`

This bug is meant to screw with discarding the hand and will show up when players choose to discard their hand to draw up to 4 cards.

- Minion bug 2: `for (j = 0; j > 4; j++) // bug 2`
  This bug will also function to cause no cards to be drawn when drawing up to 4 cards. This bug will appear when drawing after choosing to discard the hand in use of the minion.
- Ambassador bug 1: `if (choice3 == handPos) // bug 1`
  This bug will relate a third choice variable to the hand position and return true if a nonexistent choice is made, thus making the if statement virtually unreachable. This bug will show up as soon as a player uses an ambassador card.
- Ambassador bug 2: `discardCard(handPos, nextPlayer, state, 0); // bug 2`
  This bug will discard the chosen card from the next players hand as opposed to the current players hand. This bug will be introduced when a player chooses a card to discard and return 2 into the supply and to others' hands.
- Tribute bug 1: `if (state->deckCount[nextPlayer] < 0)`
  This bug essentially makes it so that the content inside this if statement is unreachable. This will become an issue if a person has 0 cards and uses a tribute action.
- Tribute bug 2: `state->deckCount[nextPlayer]++;`
  This bug will cause the player count of cards in the deck to become unsynced with the actual count of cards in the player's deck, thus leading to potential area out of bounds exceptions or "ghost" cards in a player's deck.