Bugs

Function tests:

- compare(): no detectable bugs exist
- kingdomCards(): no detectable bugs exist
- shuffle(): no detectable bugs exist
- whoseTurn(): doesn't error-check for negative player numbers. Will fail to throw an error if a player with a negative label (e.g. Player -1) is passed in.

Card tests:

- Smithy: a bug exists that increases the number of cards in the hand by 1. For instance, if 7 cards are expected to be in the hand after the Smithy has been played, there are 8 instead. A possible cause is that the Smithy card is not being discarded once played.
- Adventurer: a bug exists that decreases the number of cards in the hand by 1. The Adventurer card is supposed to add 2 treasure cards to the hand, but the tests show that it is only adding 1. In addition, the number of cards in the deck does not decrease - it is supposed to decrease by at least 2 due to the mechanics of the Adventurer. The most likely cause is that the adventurerCard function is not searching for and picking 2 treasure cards as it should.
- Village: no detectable bugs exist
- Great Hall: no detectable bugs exist

Unit Testing

compare:

- Lines executed: 70.37% of 27
- Branches executed: 57.14% of 28
- Taken at least once: 42.86% of 28
- Calls executed: 50% of 20
- All fail conditions were never executed, which is ideal. It seems that these lines are the only ones that were not executed. This test's conditionals could be cleaned up.

kingdomCards:

- Lines executed: 67.03% of 91
- Branches executed: 100% of 42
- Taken at least once: 57.14% of 42
- Calls executed: 59.46% of 37
- Fail conditions were the only commands never executed. All branches were executed.

shuffle:

- Lines executed: 74.07% of 54
- Branches executed: 69.23% of 52
- Taken at least once: 42.31% of 52
- Calls executed: 46.15% of 26
- Fail conditions were the only commands never executed. Due to imperfect branch coverage, the conditionals should be cleaned up a bit (more if-else, less if-if).

whoseTurn:

- Lines executed: 84% of 25
- Branches executed: 100% of 10
- Taken at least once: 60% of 10
- Calls executed: 75% of 16
- Some fail branches were never executed, and one success branch was never executed. Full branch execution means this test can probably remain unchanged.

Smithy:

- Lines executed: 100% of 22
- No branches
- Calls executed: 100% of 7
- I commented out the assert statements at the end because the print statements do their job better. Full statement coverage helps to indicate that the test is fine and that the Smithy card implementation is the problem. No statement is called more than once.

Adventurer:

- Lines executed: 100% of 19
- No branches
- Calls executed: 100% of 6
- Full statement coverage helps to indicate that the test is fine and that the Adventurer card implementation is the problem. No statement is called more than once.

Village:

- Lines executed: 100% of 22
- No branches
- Calls executed: 100% of 7
- Full statement coverage helps to indicate that the test is fine and that the Village card implementation is the problem. No statement is called more than once.

Great Hall

- Lines executed: 100% of 22
- No branches
- Calls executed: 100% of 7
- Full statement coverage helps to indicate that the test is fine and that the Great Hall card implementation is the problem. No statement is called more than once.

Unit Testing Efforts

compare: Since the compare function compares two integers, I decided to use several breakpoints: INT_MIN, -5, -1, 0, 1, 5, and INT_MAX. These represented every possible type of integer: minimum, negatives, zero, positive, and maximum. I used two negative and two positive integers because 1 is a common base for algorithms.
kingdomCards: I passed in 10-long arrays of similar values, ascending values, and descending values. I then tested them against their pre-modified selves to determine whether or not kingdomCards made the array properly or not.
shuffle: I tested the following cases: 0 cards, 1 card, 2 cards, and many cards. Testing 0 cards simply involved checking to see whether or not the function returned -1. All other cases involved checking to see whether the card count remained unchanged and whether the cards themselves were changed. In the "many cards" case, I also included a clause where if the resulting deck was ordered the same as the original, unshuffled deck, the test displayed a warning message to the user asking them to run the test again. While it is possible for the deck to stay as-is after a shuffle, it is highly unlikely, and the chances grow smaller as the deck grows larger.
whoseTurn: I tested all values from 0 to 4, as the player limit (MAX_PLAYERS in dominion.h) is set to 4. To see whether or not the function error-checked for negative numbers, I also tested -1, which ended up showing a small fault in the function.

Smithy: Testing Smithy simply involved checking whether the number of cards in the player's hand increased by 2 (+3 from Smithy, -1 from discarding the played Smithy card), checking that the deck count decreased by 3, then checking that the coin and action counts remained the same. There are some commented-out assert statements at the end that may be used if need be, but the print statements above them get the job done without aborting the entire program.
Adventurer: As the coin update doesn't take place in the cardEffect function, tracking the gained treasure was not a priority. As the Adventurer card puts two extra cards in the player's hand, the net gain for the hand should be 1 and the net loss from the deck should be at least 2.
Village: Just like Smithy, except instead of just testing for gained cards, the program also tests for gained actions. It also checks to make sure all unmodified variables remained the same.
Great Hall: Same as Village, but with one fewer action gain. Due to the near-identical mechanics, I was able to simply copy over the Village card test code and modify the extraActions variable.