Assignment 3

liam beckman

1 Bugs

A prominent bug is one I introduced in Assignment 2 within the `myAdventurer` function.

```
int myAdventurer(int drawntreasure, int currentPlayer, int temphand[], struct gameState
↪  *state)
{
    // ...

    // BUG: gold is not counted toward drawntreasure
    //if (cardDrawn == copper || cardDrawn == silver || cardDrawn == gold)
    if (cardDrawn == copper || cardDrawn == silver)
        drawntreasure++;

    // ...
}
```

Listing 1: myAdventurer Bug that doesn't count gold toward treasure.

2 Unit Testing

2.1 Unit Tests

The four unit tests covered the following four functions:

1. getCost
2. isGameOver
3. compare
4. updateCoins

2.1.1 unittest1 (getCost function)

```
Lines executed:1.95% of 616
```

Table 1: Coverage for Function Tests

function	statement coverage	branch coverage
getCost	1.95%	6.75%
isGameOver	1.62%	1.93%
compare	0.97%	0.96%
updateCoins	2.27%	1.93%

```
Branches executed:6.75% of 415
Taken at least once:1.20% of 415
Calls executed:0.00% of 108
```

2.1.2 unittest2 (isGameOver function)

```
Lines executed:1.62% of 616
Branches executed:1.93% of 415
Taken at least once:1.93% of 415
Calls executed:0.00% of 108
```

2.1.3 unittest3 (compare function)

```
Lines executed:0.97% of 616
Branches executed:0.96% of 415
Taken at least once:0.96% of 415
Calls executed:0.00% of 108
```

2.1.4 unittest4 (updateCoins function)

```
Lines executed:2.27% of 616
Branches executed:1.93% of 415
Taken at least once:1.69% of 415
Calls executed:0.00% of 108
```

2.2 Card Tests

The four card tests covered the following four card types:

1. smithy
2. adventurer
3. village
4. mine

Table 2: Coverage for Card Tests

card	statement coverage	branch coverage
Smithy	6.49%	3.86%
Adventurer	17.05%	17.83%
Village	19.32%	22.89%
Mine	21.27%	28.92%

2.2.1 cardtest1 (smithy)

```
Lines executed:6.49% of 616
Branches executed:3.86% of 415
Taken at least once:2.41% of 415
Calls executed:2.78% of 108
```

2.2.2 cardtest2 (adventurer)

```
Lines executed:17.05% of 616
Branches executed:17.83% of 415
Taken at least once:13.98% of 415
Calls executed:10.19% of 108
```

2.2.3 cardtest3 (village)

```
Lines executed:19.32% of 616
Branches executed:22.89% of 415
Taken at least once:13.98% of 415
Calls executed:11.11% of 108
```

2.2.4 cardtest4 (mine)

```
Lines executed:21.27% of 616
Branches executed:28.92% of 415
Taken at least once:16.63% of 415
Calls executed:12.04% of 108
```

3 Unit Testing Efforts

3.1 Function Testing

Funciton testing involved confirming that a given function had the desired effect. For example, testing the `isGameOver` function involved first confirming that a given game is still continuing, then changing a given value to trigger an end to the game (e.g. changing three supply counts to 0), and finally testing for a successful end of game value:

```
myAssert(isGameOver(&testGame) == 0);

for (i = 0; i < 3; i++)
{
    testGame.supplyCount[i] = 0;
}

myAssert(isGameOver(&testGame) == 1);
```

Listing 2: Function testing example.

3.2 Card Testing

Testing cards involved two steps:

1. Confirming that the implementation exited correctly by asserting that the relevant function returned 0 (as opposed to -1 for an unsuccessful exit).
2. Confirming that the actions of the card had the desired effect. For the following smithy example, this means that if a player started with five cards in both their hands and decks, then the smithy card would result in a deck count of two and a hand count of seven.

```
// initialize hand and deck counts
testGame->deckCount[player] = 5;
testGame->handCount[player] = 5;

// check successful exit of function
myAssert(mySmithy(player, testGame, handPos) == 0);

// check successful
myAssert(testGame->deckCount[player] == 2);
myAssert(testGame->handCount[player] == 7);
```

Listing 3: Card testing example.

As per the instructor recommendation all of the unit tests and the card tests utilized a custom assert implementation, so as to be able to collect coverage information even when a crash fails (this behavior is not included in the standard C `assert` function).

```
#define myAssert(expression)                            \
    if (expression)                                     \
        (passed)(#expression, __LINE__, __FILE__, 0); \
    else                                                \
        (failed)(#expression, __LINE__, __FILE__, 0); \

// outputs "TEST FAILED" message
void failed(char *expression, int line, char *file, int color);

// outputs "TEST PASSED" message
void passed(char *expression, int line, char *file, int color);
```

Listing 4: Custom assert macro for use in the test suite.