Class: CS-362
Term: Summer 2017
Author: Jon-Eric Cook
Date: July 30, 2017
Assignment: #5

Re-factoring

Per the instructions, a teammate's (Amelia Le) code was copied and put into a separate folder within my "projects/cookjone" folder. The name of the folder is "leameDominion". All my test code was copied pasted into leameDominion/dominion. Said test code is as follows: unittest1.c, unittest2.c, unittest3.c, unittest4.c, cardtest1.c, cardtest2.c, cardtest3.c, cardtest4.c, randomtestadventurer.c, randomtestcard1.c, and randomtestcard2.c. Once all the code was in place, the Makefile was modified to compile and run all the above files and have their outputs sent to assignment5tests.out.

Upon further inspection of Amelia's code, it was a pleasant surprise to see that she chose similar cards and functions as me to run her tests on. When it came to which cards she chose to introduce bugs into, again, there were a lot of similarities. In her play_adventurer card function, she chose to introduce a bug that borderline couldn't be caught. The bug was as follows:

if (state->deckCount[currentPlayer] < 1)
changed to
if (state->deckCount[currentPlayer] < 2).
If the program has to cycle through the entire deck to find a treasure card, it misses the last card.

This bug proved difficult to catch and sadly, it was not caught at all. I spoke with Amelia about this and she agreed that it was a hard bug to catch and that her randomtestadventurer.c only sometimes caught it.

Other than the above odd bug, all the other bugs were straightforward and easy to catch. Due to them being so straightforward, my test code was able to catch the bugs without too much refactoring. The one portion of my code that needed refactoring was for play_council_room. The bug in this function was that it skipped the first player, other than the current player, when it came time to drawing an extra card. This refactoring took place in cardtest3.c. The one change was simply making player1 the current player rather than player0. By this small refactoring, the bug in play_council_room was caught.

On the topic of maintainability, I believe my test suit is quite maintainable. When I set out to build my tests, I made sure that it would run every line of code in the FUT. By doing this, I was able to test each aspect of the function and also confirm the changes to the game state were correct. If the changes to the game state were wrong (a bug), it would be caught. Because I had this approach, I didn't have to refactor my test code a lot when it came to testing Amelia's code.

Bug-Reports

The first bug that was found occurred in the play_smithy function on line 696. This bug was found by setting up the game state and then calling cardEffect with smithy as the card argument. Once the cardEffect function returned, the hand count and deck count for each player was compared to the expected outcome. It was found that the bug caused the current

player to draw 4 cards instead of only 3. The root cause of this bug was that the for loop that was in charge of drawing the cards for the current player had its upper bound increased by one. The correct version and the buggy version of the code block can be seen below.

```
// draw 3 cards
for (i = 0; i < 3; i++)
{
	drawCard(currentPlayer, state);
}

// **BUG** draw 4 Cards
for (i = 0; i < 4; i++)
{
	drawCard(currentPlayer, state);
}
```

As a whole, I would not say this is an especially interesting bug. This type of bug could have simply been a slip of a finger, pressing 4 instead of 3.

The second bug that was found occurred in the play_council_room function on line 722. This bug was found by setting up the game state and then calling cardEffect with council_room as the card argument. Once the cardEffect function returned, the hand count and deck count for each player was compared to the expected outcome. It was found that the bug caused the program to miss player 0 when it came time for each player, other than current player, to draw a card. The root cause of this bug was that the for loop that was in charge of drawing the cards for all others players, besides the current player, had its lower bound increased by one. The correct version and the buggy version of the code block can be seen below.

```
// Each player, besides the current player, draws a card
for (i = 0; i < state->numPlayers; i++)
{
	if (i != currentPlayer)
	{
		drawCard(i, state);
	}
}

// **BUG** Each player, besides the current player, draws a card
for (i = 1; i < state->numPlayers; i++)
{
	if (i != currentPlayer)
	{
		drawCard(i, state);
	}
}
```

As a whole, I would say this is a somewhat interesting bug. This type of bug might always not show up, let alone be caught. If for example, the current player in 0, then this bug would have no effect.

The third bug that was found occurred in the play_village function on line 745. This bug was found by setting up the game state and then calling cardEffect with village as the card

argument. Once the cardEffect function returned, the number of actions for the game state was compared to the expected outcome. It was found that the bug caused the numActions count to one less than it was suppose to be. The root cause of this bug was that the variable numActions was only incremented once when it was suppose to incremented twice. The correct version and the buggy version of the code block can be seen below.

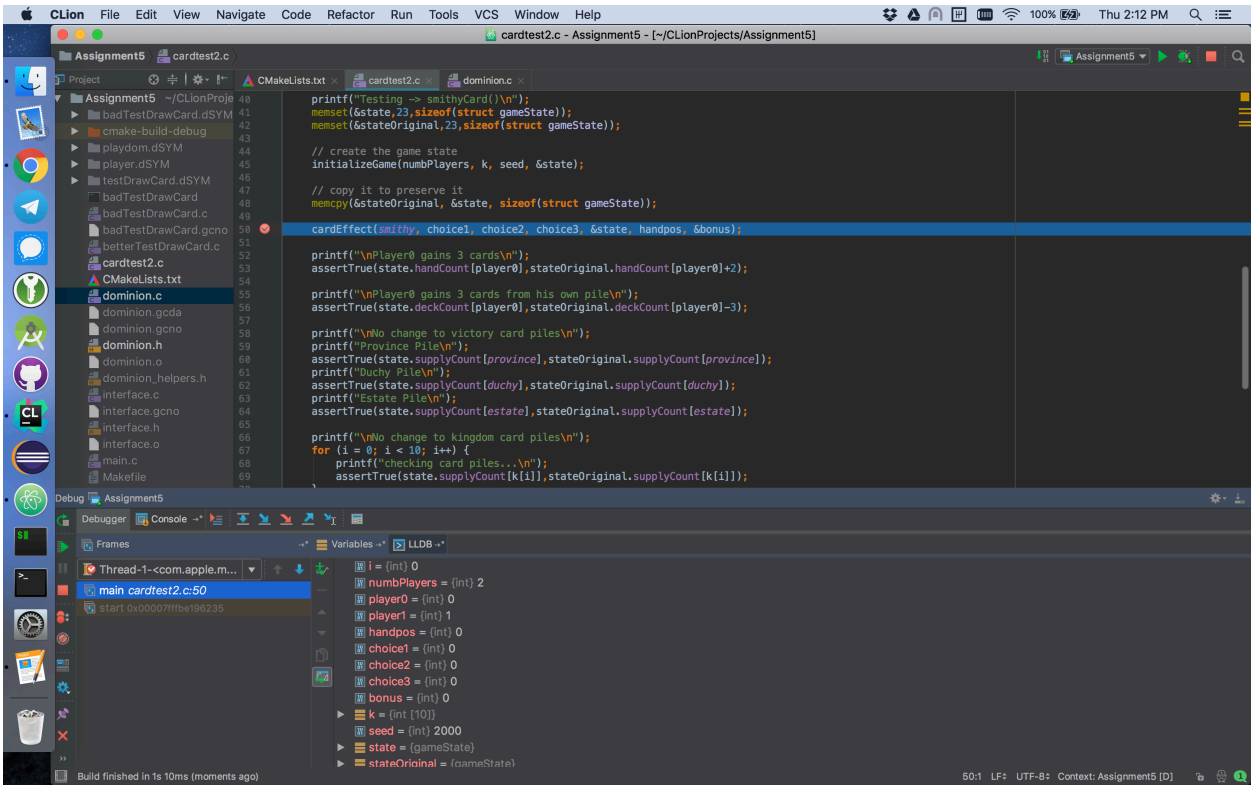
```
// +2 Actions
state->numActions = state->numActions + 2;

// **BUG** +1 Action
state->numActions = state->numActions + 1;
```

Again, as a whole, I would not say this is an especially interesting bug. This type of bug could have simply been a slip of a finger, pressing 1 instead of 2.

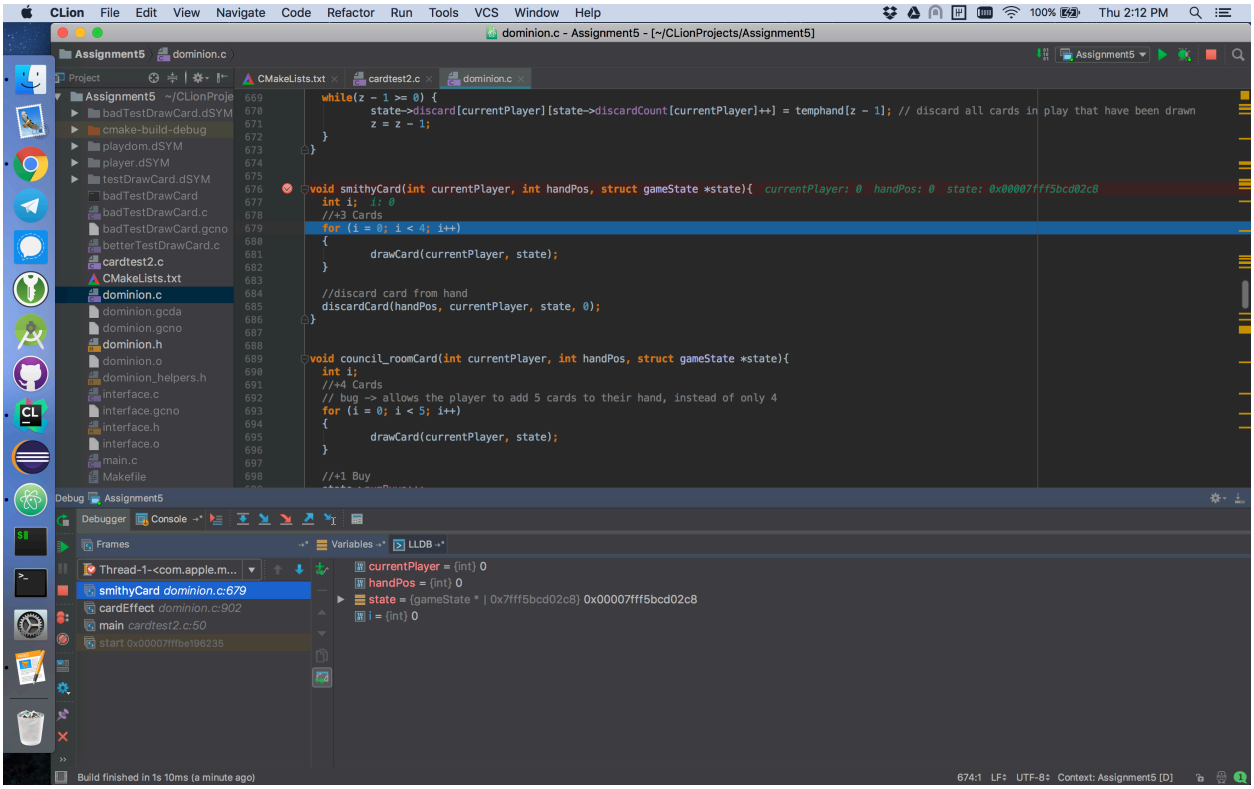
Debugging

The bug report for my code, “BugsInTeammateCode.txt”, located in projects/cookjone/dominion, was the starting place for the process of identifying and fixing a bug in my own code. From line 1 to 19, Amelia Le identifies the failure of the smithyCard function. She outlines the expected outcome of the function -> “draw 3 cards”. She also outlines the actual outcome of the function -> “draw 4 cards”. Towards the bottom of the bug report, line 18, she indicates line 680 in my dominion.c file holds the error and also provides a snippet of the bug filled code. Based on this information, I knew where to look and how to debug it. Employing the debugging tools in CLion was the next step in fixing the bug.

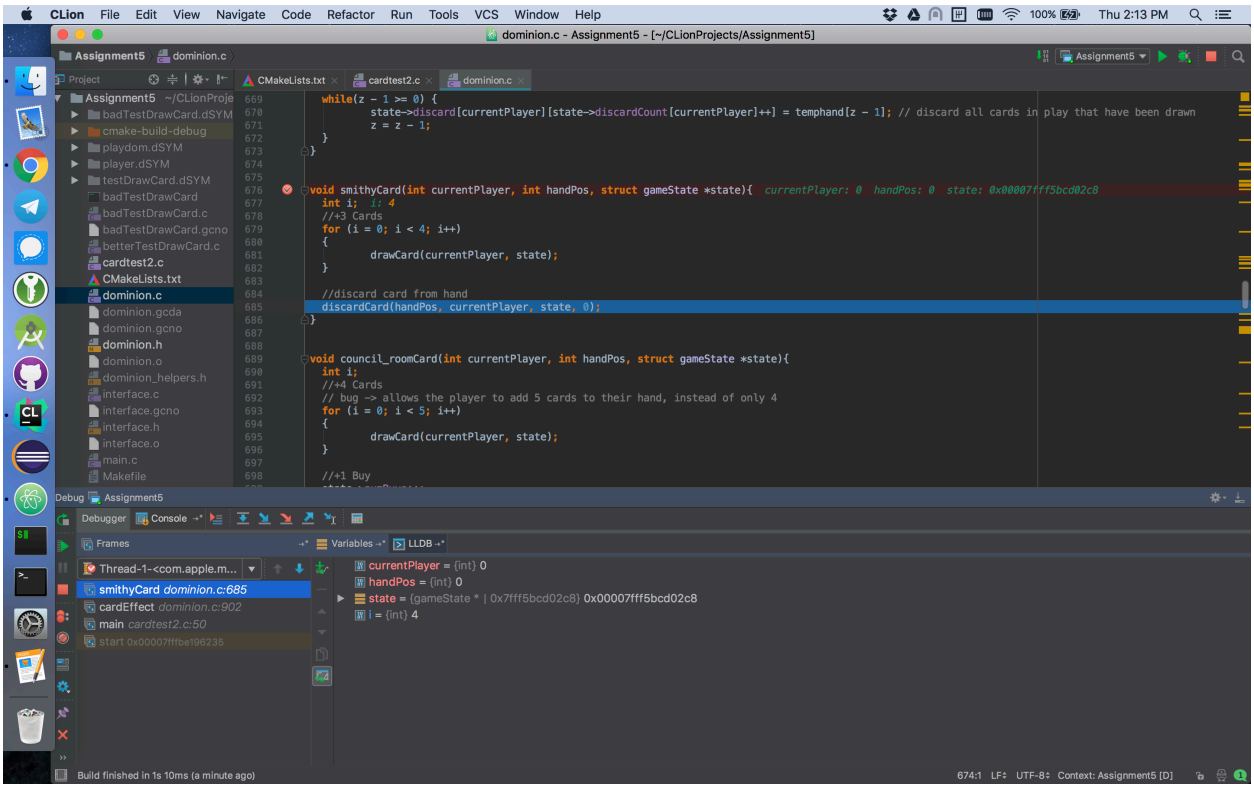
Jumping into CLion, I put break points at line 50 in cardtest2.c, and line 676 in dominion.c. These two places were chosen because they were where cardEffect was called in the cardtest2.c and where the smithyCard functions started in dominion.c. By placing break points at these two places, it allowed me to step through the bug filled code and examine what was going on, line by line. Below is the starting place of the debugging process.



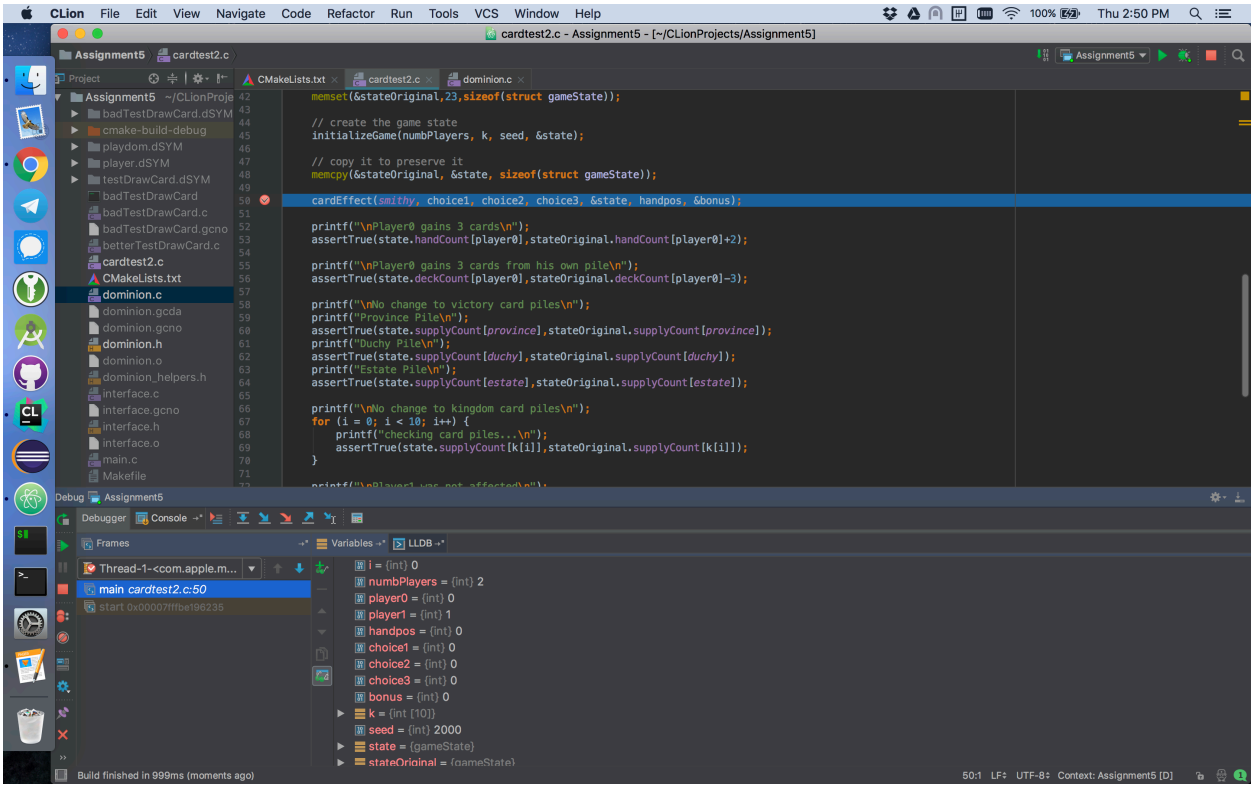
The next step was to step into the cardEffect function and make my way to the smithyCard function. Below the starting place of stepping through the smithyCard function.



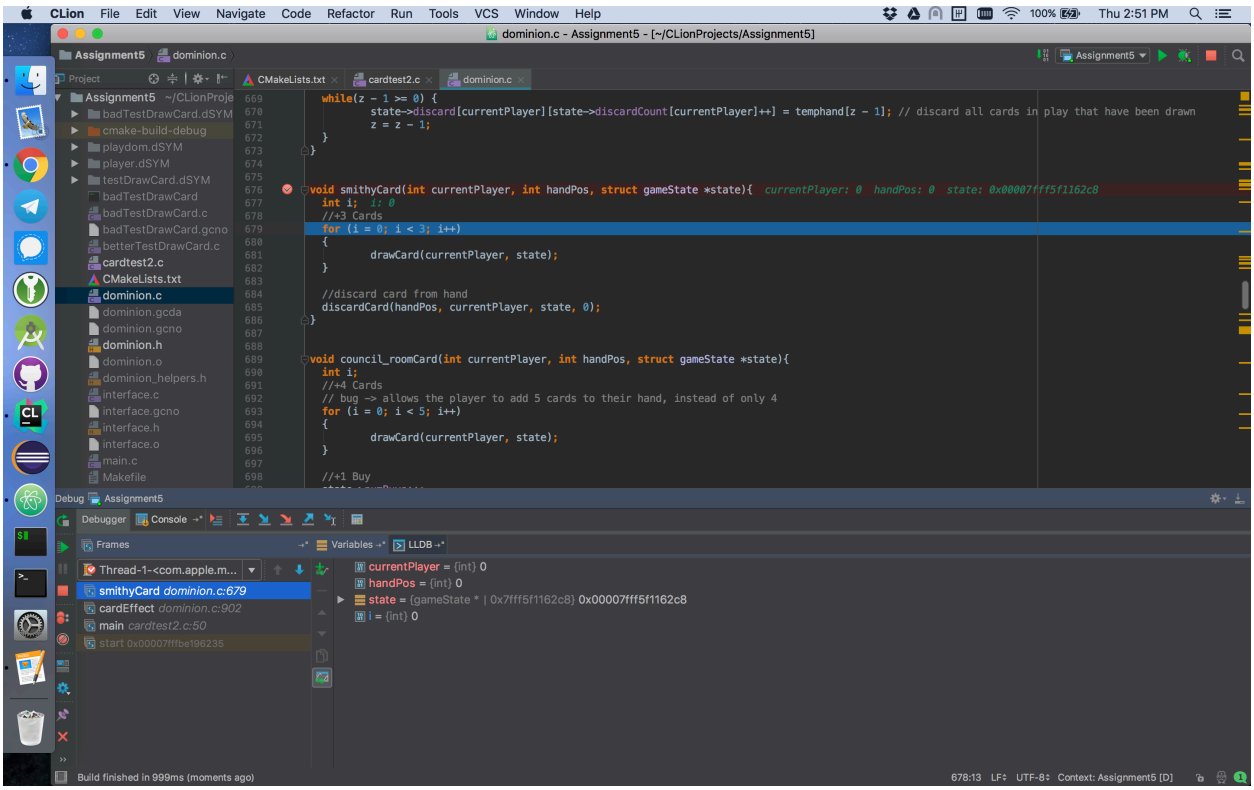
From this location I proceeded to step through the for loop until it finished. Note that it can be seen that the “i” counter starts at 0 and “should” go to only 3. Below is when the for loop and have fully cycled through.



In the above photo, it can be seen that “i” incremented all the way up to 4, meaning that the for loop ran 4 times. Thus the drawCard function was executed 4 times. This is the bug that Amelie Le pointed out. Based on the above discovery, changes were made to fix the bug. The bug was fixed by simply changing the upper bound of the for loop on line 679 in dominion.c to 3 instead of 4. A new debugging session was started and below is the start of it.



Continuing on, below it can be seen that the upper bound was in fact changed.



In the photo below, it can be seen that “i” only incremented up to 3, meaning that the for loop ran 3 times. Thus the drawCard function was executed 3 times, the correct amount.