



BUGS

It's probably helpful to outline the effects of the Assignment 2 refactor considering there are intentional bugs written into the code and thus effect the outcome of the card and unit tests.

CARD	BUG / EFFECT
great_hall_card	CLEAN function. No bugs
council_room_card	BUG: Should provide another BUY turn, but does not
village_card	BUG: Provides four (4) ACTION turns rather than two (2)
adventurer_card	BUG: Increments the drawn treasure by four (4) rather than one (1)
smithy_card	BUG: Draws four (4) cards but should only draw three (3)

The following outlines the bugs discovered during a single instance of each test being run.

TEST / FUNCTION	FAILURE
UNIT TEST 1: SHUFFLE()	Deck shuffled by shuffle() function FAILED
UNIT TEST 3: UPDATECOINS()	COINT COUNT – COPPER, Expected output: 8, Actual output: 3, FAILED
UNIT TEST 3: UPDATECOINS()	COINT COUNT – SILVER, Expected output: 13, Actual output: 3, FAILED
UNIT TEST 3: UPDATECOINS()	COINT COUNT – GOLD, Expected output: 18, Actual output: 3, FAILED
CARD TEST 1: SMITHY	FAIL: Incorrect HAND COUNT after playing SMITHY. Expected output 7, actual output 8
CARD TEST 1: SMITHY	FAIL: Incorrect DECK COUNT after playing SMITHY. Expected output 2, actual output 1
CARD TEST 1: SMITHY	FAIL: Incorrect SCORE after playing SMITHY. Expected output 1, actual output 2
CARD TEST 2: ADVENTURER	FAIL: Incorrect DECK COUNT after playing ADVENTURER. Expected output 3, actual output 4
CARD TEST 3: COUNCIL ROOM	PLAYER-1 number of buys increments FAILED
CARD TEST 4: VILLAGE	FAIL: Incorrect ACTIONS count after playing VILLAGE. Expected output 3, actual output 5

For the most part, the test failures align with the known/manufactured bugs, however the updatesCoins() unit test could be an issue with the test logic, an issue with the card being used to test the function (in this case, it would make sense that the test would fail if ADVENTURER is played, and it is the first in the available testCards array), or an actual bug within the function. At the very least, the test outlines an actionable set of troubleshooting tasks that, if followed, should alleviate the issue(s).

UNIT TESTING

I'll start by saying I took a test suite approach to summarize the impact of all the tests, that is, I intentionally did not remove the .gcda file between each gcov execution in the MAKEFILE. I took this approach because it seemed like from an outsider's perspective, the most likely scenario would be that s/he would run all the available tests to get a holistic perspective on existing test coverage as a means to hunt-down a bug or continue to increase the code base' coverage. This cumulative approach is also



significantly easier to read; a point worth considering when specificity can actually be a hindrance to targeting a problem.

This said, in the process of building the MAKEFILE, and ultimately in the process of building the individual tests, I did build it so that it could produce a non-cumulative output file. By removing the .gcda file between every gcov execution, it effectively exposes each test's strengths and weaknesses, and eliminates cross-coverage obfuscation. Of course, the downside to this is that the output file is excessively long, making manual review (and especially test coverage comparisons) especially time consuming. You can execute this by running the `$ make exhaustiveTests` which outputs to `exhaustiveunittestresults.out`

For clarification, the following results are from the `unittestresults.out` file, and are a cumulative summary of the effect of all existing tests. The `gcov -b` (branch probabilities) and `-f` (function summaries) flags were set in the `runtests` MAKEFILE target. Additionally the `dominion.c.gov` file is concatenated to the end of the `unittestresults.out` file.

GCOV BRANCH PROBABILITIES & FUNCTION SUMMARIES

Function 'updateCoins'

Lines executed:100.00% of 11
Branches executed:100.00% of 8
Taken at least once:100.00% of 8
No calls

Function 'gainCard'

Lines executed:100.00% of 13
Branches executed:100.00% of 6
Taken at least once:100.00% of 6
Calls executed:100.00% of 1

Function 'discardCard'

Lines executed:84.62% of 13
Branches executed:100.00% of 6
Taken at least once:50.00% of 6
No calls

Function 'cardEffect'

Lines executed:5.37% of 205
Branches executed:12.85% of 179
Taken at least once:3.35% of 179
Calls executed:8.62% of 58

Function 'smithy_card'

Lines executed:100.00% of 6
Branches executed:100.00% of 2
Taken at least once:100.00% of 2
Calls executed:100.00% of 3

Function 'adventurer_card'

Lines executed:100.00% of 18
Branches executed:100.00% of 12
Taken at least once:100.00% of 12
Calls executed:100.00% of 3

Function 'council_room_card'

Lines executed:100.00% of 9
Branches executed:100.00% of 6
Taken at least once:100.00% of 6
Calls executed:100.00% of 4

Function 'village_card'

Lines executed:100.00% of 6
No branches
Calls executed:100.00% of 3

Function 'great_hall_card'

Lines executed:0.00% of 6
No branches
Calls executed:0.00% of 3

Function 'getCost'

Lines executed:23.33% of 30
Branches executed:100.00% of 28
Taken at least once:17.86% of 28
No calls

Function 'drawCard'

Lines executed:95.45% of 22
Branches executed:100.00% of 6
Taken at least once:83.33% of 6
Calls executed:100.00% of 1

Function 'getWinners'

Lines executed:0.00% of 24
Branches executed:0.00% of 22
Taken at least once:0.00% of 22
Calls executed:0.00% of 2

Function 'scoreFor'

Lines executed:100.00% of 24
Branches executed:100.00% of 42
Taken at least once:69.05% of 42
Calls executed:0.00% of 3

Function 'isGameOver'

Lines executed:100.00% of 10
Branches executed:100.00% of 8
Taken at least once:100.00% of 8
No calls

Function 'endTurn'

Lines executed:100.00% of 20
Branches executed:100.00% of 6
Taken at least once:100.00% of 6
Calls executed:100.00% of 3

Function 'whoseTurn'

Lines executed:100.00% of 2
No branches
No calls

Function 'fullDeckCount'

Lines executed:0.00% of 9
Branches executed:0.00% of 12
Taken at least once:0.00% of 12
No calls

Function 'supplyCount'

Lines executed:100.00% of 2
No branches
No calls

Function 'handCard'

Lines executed:100.00% of 3
No branches
Calls executed:100.00% of 1

Function 'numHandCards'

Lines executed:100.00% of 2
No branches
Calls executed:100.00% of 1



DREW WOLFE | ONID: WOLFEDR
CS 362-400 WINTER 2018
ASSIGNMENT: 3

Function 'buyCard' Lines executed:76.92% of 13 Branches executed:100.00% of 6 Taken at least once:50.00% of 6 Calls executed:100.00% of 4	Calls executed:100.00% of 2	No calls
Function 'playCard' Lines executed:78.57% of 14 Branches executed:100.00% of 10 Taken at least once:60.00% of 10 Calls executed:100.00% of 3	Function 'initializeGame' Lines executed:83.87% of 62 Branches executed:95.65% of 46 Taken at least once:80.43% of 46 Calls executed:100.00% of 5	Function 'compare' Lines executed:100.00% of 6 Branches executed:100.00% of 4 Taken at least once:100.00% of 4 No calls
Function 'shuffle' Lines executed:100.00% of 16 Branches executed:100.00% of 8 Taken at least once:100.00% of 8	Function 'kingdomCards' Lines executed:0.00% of 13 No branches No calls	File 'dominion.c' Lines executed:48.22% of 562 Branches executed:53.96% of 417 Taken at least once:36.93% of 417 Calls executed:39.00% of 100
	Function 'newGame' Lines executed:0.00% of 3 No branches	

The highlighted functions are those that were covered in Assignment 3. It's generally satisfying to know that they all have 100% line and branch coverage. As mentioned above in the BUGS section, I cannot confirm that the coverage is 100% accurate, but it's a start. The findings are particularly nice in the sense that they make coverage shortages explicit and specific. I know exactly what functions could use additional coverage and what functions are, at least as it relates to blanket coverage, covered. I didn't take the time to determine whether each function had open boundaries or, if not, what those boundary values are, so I cannot speak to the boundary test coverage. I used pre-defined MAX and MIN values in a couple tests, but I hardly think this can be considered adequate boundary testing.

The concatenated output of `dominion.c.gcov` with branch probabilities appears to be where a fair portion of the gcov value rests. A simple `cmd + f` to find "fallthrough" accompanied by a relatively high percentage seems to be a fruitful means to find bug opportunities.



DREW WOLFE | ONID: WOLFEDR
CS 362-400 WINTER 2018
ASSIGNMENT: 3

```
dominion.c
85 //set number of Victory cards
86 if (numPlayers == 2)
87 {
88     state->supplyCount[estate] = 8;
89     state->supplyCount[duchy] = 8;
90     state->supplyCount[province] = 8;
91 }
92 else
93 {
94     state->supplyCount[estate] = 12;
95     state->supplyCount[duchy] = 12;
96     state->supplyCount[province] = 12;
97 }
98
99 //set number of Treasure cards
100 state->supplyCount[copper] = 60 - (7 * numPlayers);
101 state->supplyCount[silver] = 40;
102 state->supplyCount[gold] = 30;
103
104 //set number of Kingdom cards
105 for (i = adventurer; i <= treasure_map; i++) //loop all cards
106 {
107     for (j = 0; j < 10; j++) //loop chosen cards
108     {
109         if (kingdomCards[j] == i)
110         {
111             //check if card is a 'Victory' Kingdom card
112             if (kingdomCards[j] == great_hall || kingdomCards[j] == gardens)
113             {
114                 if (numPlayers == 2){
115                     state->supplyCount[i] = 8;
116                 }
117                 else{ state->supplyCount[i] = 12; }
118             }
119             else
120             {
121                 state->supplyCount[i] = 10;
122             }
123             break;
124         }
125     }
126     else //card is not in the set chosen for the game
127     {
128         state->supplyCount[i] = -1;
129     }
130 }
131
132 //supply initialization complete
133
134 //set player decks
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

UNIT TESTING EFFORTS

I've written very simple inline unit tests for a small handful of larger projects in the past, but those were written largely as a means to eliminate the sources of a larger bug/problem. While I initially began writing the tests with the intention of providing the best possible coverage, it was only after I began running gcov, and after re-watching the lectures on random testing, (and after speaking with some classmates that had far more complex testing conventions that I employed) that I realized that I could be doing a significantly better job. That said, for an initial effort, I think it was a reasonable effort and an exceptionally eye-opening experience.

The basic approach I took for the function tests was to break down each function's objective and determine whether completing the objective could be influenced by parameters the function accepted. I generally wrote individual tests that made comparisons between a previous game state and the a game state influenced by the function. I would add or subtract a given struct attribute to the function-effected game state as a means to create an equality between the two states. Knowing that the addition or subtraction of the struct attributes were static, or hard-coded, I knew that if the game states varied, then something went wrong with the function as a result of a specific input/parameter.

I could have been more thorough, especially in hindsight, but the underlying approach, of which I obviously cannot take credit, proved to be sound. It generally took the same approach toward the card tests as well.