How to Write a Simple Random Tester

Building a Simple Random Tester

1. Identify the interface to test
   - Is it a file interface?
   - Network interface?
   - Calls to a function?
2. Write code to generate random inputs
   - What values is the code expected to handle?
   - Are all of these values interesting?
3. Write code to check behavior on random inputs
   - How can you tell if it worked?

Recipe for Refining a Random Tester

1. Gather code coverage
   - Is everything interesting being covered?
   - Is important code not covered?
2. Adjust the code to generate inputs
   - Try to “stay random” but shift the probability space
   - Augment random with fixed inputs of interest
3. Break the code and see if your tests detect the problem
   - If not, why not?
4. Improve your oracle code until all problems that should be caught *are* caught
5. Repeat until coverage and “fake bugs” both show the testing is rock solid

/cygdrive/c/Documents and Settings/Alex/Desktop/cs362class/dominion

File Edit Options Buffers Tools C Help

```
int drawCard(int player, struct gameState *state)
{
  int count;
  int deckCounter;
  if (state->deckCount[player] <= 0){//Deck is empty
    int i;
    for (i = 0; i < state->discardCount[player];i++){
      state->deck[player][i] = state->discard[player][i];
      state->discard[player][i] = -1;
    }

    state->deckCount[player] = state->discardCount[player];
    state->discardCount[player] = 0;//Reset discard

    shuffle(player, state);//Shuffle the deck up and make it so that we can draw

    if (DEBUG){//Debug statements
      printf("Deck count now: %d\n", state->deckCount[player]);
    }

    state->discardCount[player] = 0;

    count = state->handCount[player];//Get current player's hand count

    if (DEBUG){//Debug statements
      printf("Current hand count: %d\n", count);
    }

    deckCounter = state->deckCount[player];//Create a holder for the deck count

    if (deckCounter == 0)
      return -1;

    state->hand[player][count] = state->deck[player][deckCounter - 1];//Add card to hand
    state->deckCount[player]--;
    state->handCount[player]++;//Increment hand count
  }

  else{
    int count = state->handCount[player];//Get current hand count for player
    int deckCounter;
    if (DEBUG){//Debug statements
      printf("Current hand count: %d\n", count);
    }

    deckCounter = state->deckCount[player];//Create holder for the deck count
    state->hand[player][count] = state->deck[player][deckCounter - 1];//Add card to the hand
    state->deckCount[player]--;
    state->handCount[player]++;//Increment hand count
  }

  return 0;
}
```

-UU-:**--F1 dominion.c 38% L548 (C/l Abbrev)----------

What code are we testing?

```
int drawCard(
    int player,
    struct gameState *state
);
```

What inputs does it take?

```
int drawCard(
    int player,
    struct gameState *state
);
```

Can we randomly generate these inputs?

Yes…

/cygdrive/c/Documents and Settings/Alex/Desktop/cs362class/dominion

File Edit Options Buffers Tools C Help

```
int main () {

  int i, n, r, p, deckCount, discardCount, handCount;

  int k[10] = {adventurer, council_room, feast, gardens, mine,
               remodel, smithy, village, baron, great_hall};

  struct gameState G;

  printf ("Testing drawCard.\n");

  printf ("RANDOM TESTS.\n");

  SelectStream(2);
  PutSeed(3);

  for (n = 0; n < 2000; n++) {
    for (i = 0; i < sizeof(struct gameState); i++) {
      ((char*)&G)[i] = floor(Random() * 256);
    }
    p = floor(Random() * 1000);
    checkDrawCard(p, &G);
  }

  printf ("ALL TESTS OK\n");

  exit(0);
}
```

-UU-:**--F1 badTestDrawCard.c   Bot L54   (C/l Abbrev)----------

This code is generating random tests:

1. Create a gameState G filled with random bytes
2. Choose a number of players randomly
3. Call a function to test drawCard with these inputs

/cygdrive/c/Documents and Settings/Alex/Desktop/cs362class/dominion

```
File Edit Options Buffers Tools C Help
int main () {

  int i, n, r, p, deckCount, discardCount, handCount;

  int k[10] = {adventurer, council_room, feast, gardens, mine,
               remodel, smithy, village, baron, great_hall};

  struct gameState G;

  printf ("Testing drawCard.\n");

  printf ("RANDOM TESTS.\n");

  SelectStream(2);
  PutSeed(3);

  for (n = 0; n < 2000; n++) {
    for (i = 0; i < sizeof(struct gameState); i++) {
      ((char*)&G)[i] = floor(Random() * 256);
    }
    p = floor(Random() * 1000);
    checkDrawCard(p, &G);
  }

  printf ("ALL TESTS OK\n");

  exit(0);
}

-UU-:**--F1  badTestDrawCard.c    Bot L54     (C/l Abbrev)----------------
```

What happens when we run this tester?

/cygdrive/c/Documents and Settings/Alex/Desktop/cs362class/dominion

```
Alex@groce /cygdrive/c/Documents and Settings/Alex/Desktop/cs362class/dominion
$ ./badTestDrawCard.exe
Testing drawCard.
RANDOM TESTS.
Segmentation fault (core dumped)

Alex@groce /cygdrive/c/Documents and Settings/Alex/Desktop/cs362class/dominion
$
```

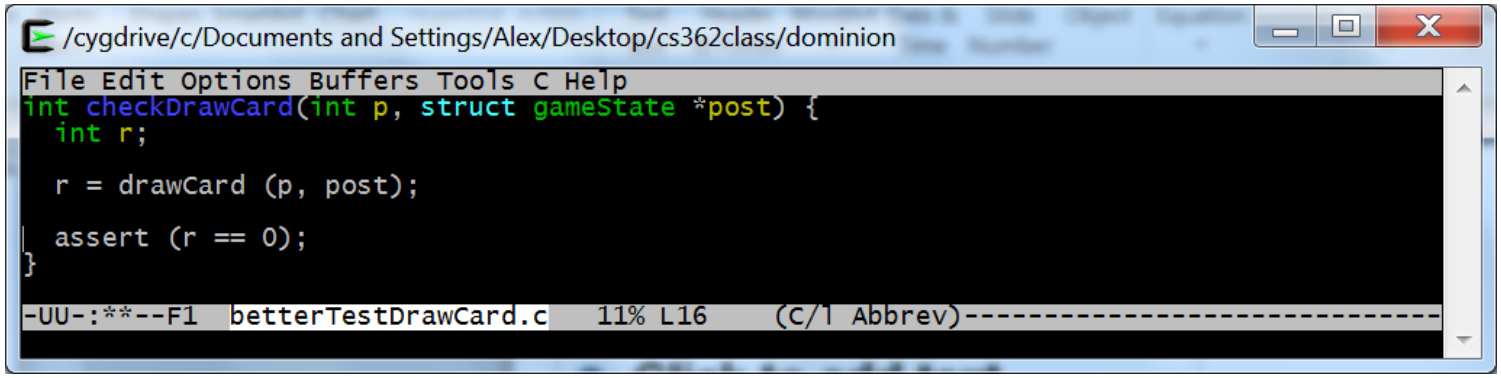
Pure Random Seldom Works!

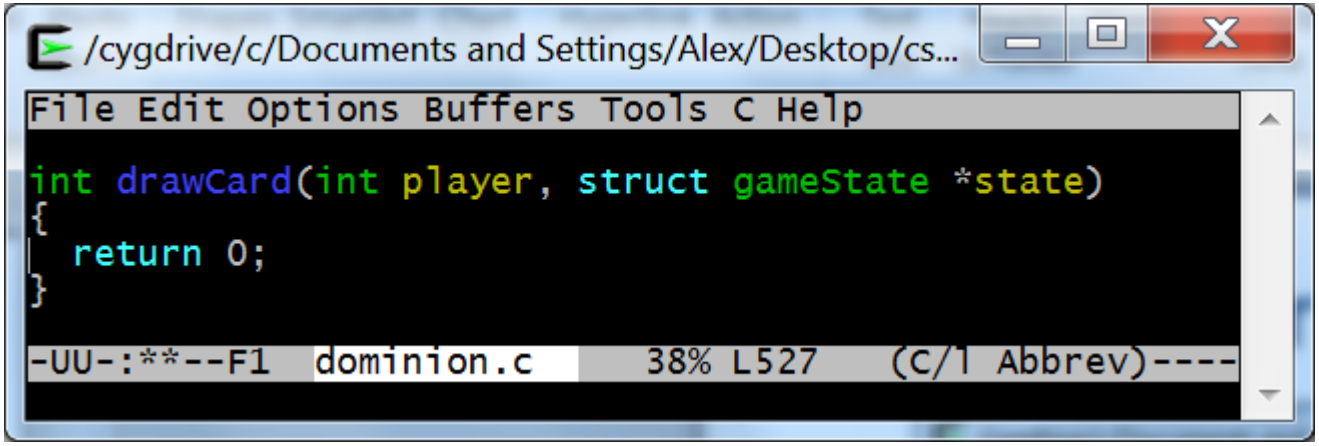
- **Need to think about preconditions**
  - drawCard expects
    - a valid number of players
    - a somewhat “sane” gameState
  - Can we generate that?

/cygdrive/c/Documents and Settings/Alex/Desktop/cs362class/dominion

```
File Edit Options Buffers Tools C Help
  SelectStream(2);
  PutSeed(3);

  for (n = 0; n < 2000; n++) {
    for (i = 0; i < sizeof(struct gameState); i++) {
      ((char*)&G)[i] = floor(Random() * 256);
    }
    p = floor(Random() * 2);
    G.deckCount[p] = floor(Random() * MAX_DECK);
    G.discardCount[p] = floor(Random() * MAX_DECK);
    G.handCount[p] = floor(Random() * MAX_HAND);
    checkDrawCard(p, &G);
  }

  printf ("ALL TESTS OK\n");

  exit(0);

  printf ("SIMPLE FIXED TESTS.\n");
-UU-:----F1  testDrawCard.c    59% L64     (C/l Abbrev)----------------------------------------
```

Check Your Test Oracle





Revise Your Test Oracle

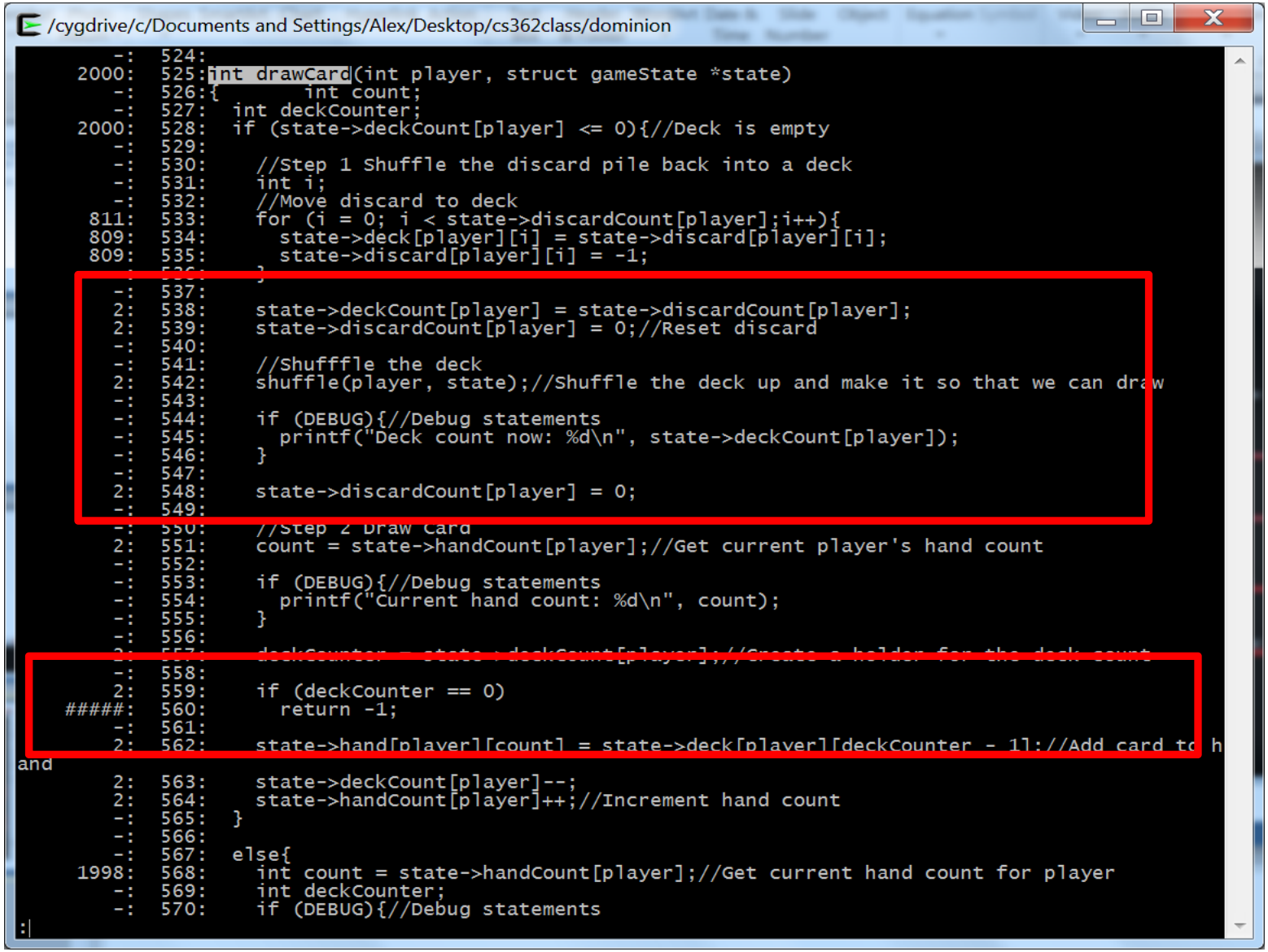
/cygdrive/c/Documents and Settings/Alex/Desktop/cs362class/dominion

File Edit Options Buffers Tools C Help

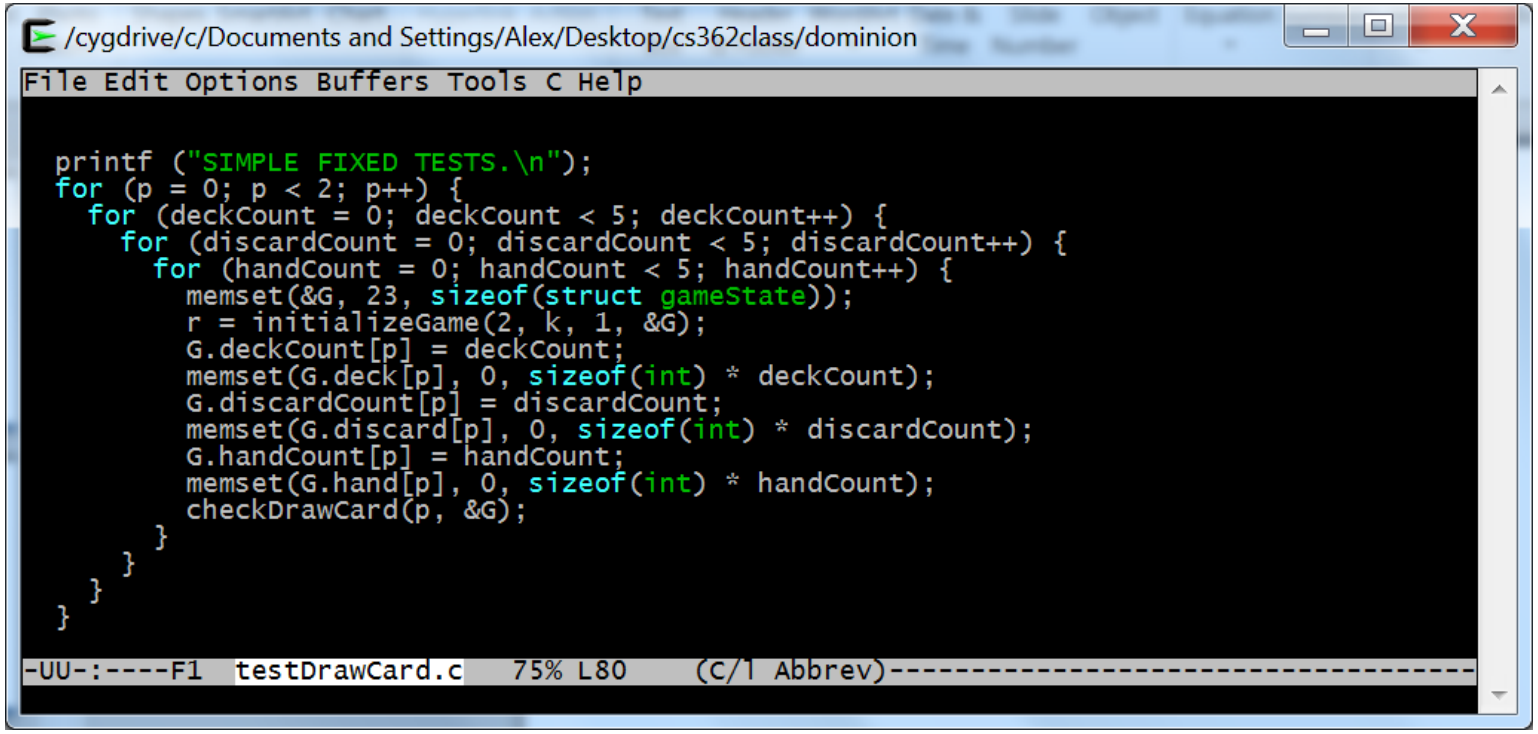
```
int checkDrawCard(int p, struct gameState *post) {
  struct gameState pre;
  memcpy (&pre, post, sizeof(struct gameState));

  int r;
  //  printf ("drawCard PRE: p %d HC %d DeC %d DiC %d\n",
  //      p, pre.handCount[p], pre.deckCount[p], pre.discardCount[p]);

  r = drawCard (p, post);

  //printf ("drawCard POST: p %d HC %d DeC %d DiC %d\n",
  //      p, post->handCount[p], post->deckCount[p], post->discardCount[p]);

  if (pre.deckCount[p] > 0) {
    pre.handCount[p]++;
    pre.hand[p][pre.handCount[p]-1] = pre.deck[p][pre.deckCount[p]-1];
    pre.deckCount[p]--;
  } else if (pre.discardCount[p] > 0) {
    memcpy(pre.deck[p], post->deck[p], sizeof(int) * pre.discardCount[p]);
    memcpy(pre.discard[p], post->discard[p], sizeof(int)*pre.discardCount[p]);
    pre.hand[p][post->handCount[p]-1] = post->hand[p][post->handCount[p]-1];
    pre.handCount[p]++;
    pre.deckCount[p] = pre.discardCount[p]-1;
    pre.discardCount[p] = 0;
  }

  assert (r == 0);

  assert(memcmp(&pre, post, sizeof(struct gameState)) == 0);
}
```

-UU-:**--F1 testDrawCard.c   7% L30   (C/l Abbrev)----------

Check Coverage



Add Fixed Tests if Needed



Final testDrawCard.c gets coverage of every line of drawCard, including the empty-deck and discard case (10 times). You can also get this result by going from 2,000 tests to 2,000,000 – which covers the empty case 23 times and only takes half an hour to run. You also get more coverage of everything else.

Don’t Work Smarter, Just Work Harder?