Zane Koch

404984675

The most notable obstacle I overcame was building the complete and evaluate functions in class board. This was hard because it required me to understand many different function within the other various classes some of which I had not even written. After going through and making sense of all the functions I realized that using isValueOk to test if certain cells values were ok made the most sense. I then realized that if I made evaluate then I could just use evaluate inside of completed because if evaluate is equal to 100 then it is completed and otherwise it is not. I then made two loops, one for columns and one for rows, that iterating through each row and column and got put into the mcell variable which I then put into isvalueOk. This worked and I was happy.

A secondary, more general, obstacle was that it was hard to understand how the 4 different classes fit together. I read the spec and eventually understood the hierarchy of numberset, cell, board and game.

NumberSet objject(1, 2, 3, 4, 5, 6, 7, 7);

NumberSet a(1, 2, 3, 4, 5, 6, 7, 7);

NumberSet b(1, 2, 3, 4, 5, 6, 7, 7);

assert(objject.stringify() == "SET-12345677");

The test above checks to see if strinify prints out the numbers of the numberset that is passed in.

assert(objject.notFound(8) == true);

assert(objject.notFound(1) == false);

assert(objject.notFound(2) == false);

The test cases above check if the notfound function is working by passing it sets of different combinations of numbers. It gives some cases with repeated values and some with non repeated values to see if it knows the correct action for each and if can identify when a value is missing.

Cell q;

NumberSet d( 1, 1, 1, 1, 1, 1, 1, 1);

NumberSet e(1, 1, 1, 1, 1, 1, 1, 1);

assert(q.isValueOK(0, objject, a, b) == true);

assert(q.isValueOK(1, objject, a, b) == false);

assert(q.isValueOK(2, objject, d, e) == false);

The test cases above check if for a given value, and three numbersets, if the value can be put into the spot where these 3 numbersets converge. The first should return true because 0 always should be able to fit. The second should return false because there is already a 1 in atleast one of the numbersets. The third assert should fail because there is already a 2 in numberset object.

NumberSet ns(1, 2, 3, 4, 5, 6, 7, 8);

assert(ns.notFound(0));

assert(!ns.notFound(1));

assert(!ns.notFound(2));

assert(!ns.notFound(3));

assert(!ns.notFound(4));

assert(!ns.notFound(5));

assert(!ns.notFound(6));

assert(!ns.notFound(7));

assert(!ns.notFound(8));

assert(ns.notFound(9));

The above asserts check if a certain number is found in a number set. The numberset it is testing gainst have number 1-8 so all asserts that have numbers 1-8 as the argument should fail—this is the case. All asserts that do not have 1-9 should not fail.

Cell c;

c.setValue(1);

assert(c.getValue() == 1);

assert(!c.isImmutable());

c.setValue(2);

c.setImmutable(true);

assert(c.getValue() == 2);

assert(c.isImmutable());

The above asserts check if set value, get value, and is immutable work. It does so by using a cell c. It then tests if inside this cell a value can be set to 1 and then to 2.

assert(c.isValueOK(0, ns, ns, ns));

assert(!c.isValueOK(1, ns, ns, ns));

assert(!c.isValueOK(2, ns, ns, ns));

assert(!c.isValueOK(3, ns, ns, ns));

assert(!c.isValueOK(4, ns, ns, ns));

assert(!c.isValueOK(5, ns, ns, ns));

assert(!c.isValueOK(6, ns, ns, ns));

assert(!c.isValueOK(7, ns, ns, ns));

assert(!c.isValueOK(8, ns, ns, ns));

assert(c.isValueOK(9, ns, ns, ns));

The above asserts are more tests for isValueOk. They each have the name numberset wjich is numbers 1-8 and so every number besides 9 and 0 should fail.

assert(!c.isOKToSetValue(0, ns, ns, ns));

assert(!c.isOKToSetValue(1, ns, ns, ns));

assert(!c.isOKToSetValue(2, ns, ns, ns));

assert(!c.isOKToSetValue(3, ns, ns, ns));

assert(!c.isOKToSetValue(4, ns, ns, ns));

assert(!c.isOKToSetValue(5, ns, ns, ns));

assert(!c.isOKToSetValue(6, ns, ns, ns));

assert(!c.isOKToSetValue(7, ns, ns, ns));

assert(!c.isOKToSetValue(8, ns, ns, ns));

assert(!c.isOKToSetValue(9, ns, ns, ns));

The assets above do the same thing but with checking for immutability aswell. No assert should return true because all the cells in numberset ns are not immutable.

Game g;

g.cheat("123456789|.........|987321456|456987123|.........|312645978|.........|.........|.........");

assert(g.display() == "123|456|789\n...|...|...\n987|321|456\n-----------\n456|987|123\n...|...|...\n312|645|978\n-----------\n...|...|...\n...|...|...\n...|...|...\n");

assert(g.evaluate() == 36);

assert(!g.completed());

assert(!g.acceptValue(0, 1, 1));

assert(!g.acceptValue(7, 1, 1));

The assert above check for a given board if evaluate and complete work. The first evaluate asserts should be true because 36 of the 81 cells are filled out correctly. The next assert should be false because 36 is not all the cells so the board is not completed. The acceptValue asserts should both be false. The last evaluate should also be