1. A brief description of notable obstacles you overcame.

The main obstacle that I faced was to program the function of player::whatSpotIsNeededNext as I considered it to be very important. My understanding of the importance of this is that I could update what has been rolled and what is needed next after one has rolled (or assigned dies by cheating). I realized this functionality by adding a private function of updateHasRolled in the player class which takes up the values of what has been just rolled with one trial (meaning the value of each die and the summation of two or three dies), and using this function to determine the private bool variables hasRolledX (hasRolled1, hasRolled2, etc.). In this way, updating the hasRolledX after each trial of rolling dies. Then to find out the whatSpotIsNeededNext, which is very crucial, I used an array to take up the updated value of hasRolledX, and use a for loop to find out the first value changing from true to false, which is the index (actually, index+2) for the next value needed. The player::rolled function, which was used for testing, was programmed with switch statement to update the hasRolledX and making sure the continuity needed.

Then, in the centennial class, I used the whatSpotIsNeededNext to keep updating the position of the player on the Board, and the setGameOver and markHuman(or computer)AsWinner to tell the board if the game is over and who is the winner.

2. A list of the test data that could be used to thoroughly test your functions, along with the reason for each test.

// Die test code

int i, value;

Die d, d1, d2, d3, d4, d5, d6;

for (i = 1; i <= 50; i++)

{

d.roll();

value = d.getValue();

assert(value >= 1 && value <= 6); // to test if rolled values are valid (meaning within 1 to 6)

}

// Player test code

Player p, human, computer;

// in the beginning of time, nothing has been rolled yet and the spot needed is 1...

assert(!p.hasRolledOne());

assert(!p.hasRolledTwo());

assert(!p.hasRolledThree());

assert(!p.hasRolledFour());

assert(!p.hasRolledFive());

assert(!p.hasRolledSix());

assert(!p.hasRolledSeven());

assert(!p.hasRolledEight());

assert(!p.hasRolledNine());

assert(!p.hasRolledTen());

assert(!p.hasRolledEleven());

assert(!p.hasRolledTwelve());

assert(p.whatSpotIsNeededNext() == 1); // before rolling, nothing is rolled and nextNeeded should be 1

// now the player has rolled 1... so the spot next needed is 2...

p.rolled(1); // letting the rolled to be 1

assert(p.hasRolledOne()); // testing hasRolled1 to be true

assert(!p.hasRolledTwo());

assert(p.whatSpotIsNeededNext() == 2); // updated as nextNeeded should be 2

// only rolls from 1-12 are relevant...

p.rolled(100);

assert(p.hasRolledOne());

assert(!p.hasRolledTwo());

assert(p.whatSpotIsNeededNext() == 2); // invalid rolled number, won’t start

// rolls must be sequential for things to count...

p.rolled(3);

assert(p.hasRolledOne());

assert(!p.hasRolledTwo());

assert(!p.hasRolledThree());

assert(p.whatSpotIsNeededNext() == 2); // can’t skip 2 to rolled 3

// work the Player via Dies

d1.setValue(6);

d2.setValue(5);

d3.setValue(4);

p.roll(d1, d2, d3);

assert(p.whatWasRolled() == "Die1: 6 Die2: 5 Die3: 4\n"); // forcing rolled value with given value of dies

// Board test code

Board b;

assert(b.getHumanSpot() == 0);

assert(b.getComputerSpot() == 0); // starting from 0 for both players

assert(b.isGameOver() == false); // when started, game is not over

assert(b.isHumanWinner() == false); // when started, no one is winner

b.setHumanSpot(3);

b.setComputerSpot(6);

assert(b.getHumanSpot() == 3);

assert(b.getComputerSpot() == 6); // testing set function

assert(b.isGameOver() == false);

assert(b.isHumanWinner() == false); // game is not over and no one wins so far

b.setHumanSpot(12);

assert(b.getHumanSpot() == 12);

assert(b.getComputerSpot() == 6);

assert(b.isGameOver() == false);

assert(b.isHumanWinner() == false); // board itself cannot determine if game is finished or not even someone has set to be at 12

b.setGameOver(true);

b.markHumanAsWinner();

assert(b.isGameOver() == true);

assert(b.isHumanWinner() == true); // board can only tell if game is over and the winner by being told with the set and mark function, changing the mGameOver and mHumanWon.

// Centennial test code

Centennial game;

assert(game.isGameOver() == false);

assert(game.determineGameOutcome() == Centennial::GAMENOTOVER);

human = game.getHuman();

computer = game.getComputer();

assert(human.whatSpotIsNeededNext() == 1);

assert(computer.whatSpotIsNeededNext() == 1); // before starting, game is not over, next spot needed should be 1.

d1.setValue(1);

d2.setValue(2);

d3.setValue(3);

d4.setValue(4);

d5.setValue(5);

d6.setValue(6);

game.humanPlay(d6, d5, d4);

human = game.getHuman();

assert(human.whatSpotIsNeededNext() == 1); // didn’t start with 6,5,4 as no 1 has been rolled

game.computerPlay(d1, d2, d3);

computer = game.getComputer();

assert(computer.whatSpotIsNeededNext() == 7); // 1,2,3 should give a nextNeeded to be 7

game.computerPlay(d5, d2, d1);

computer = game.getComputer();

assert(computer.whatSpotIsNeededNext() == 9); // nextNeeded should memorize what has been rolled

game.humanPlay(d4, d2, d1);

human = game.getHuman();

assert(human.whatSpotIsNeededNext() == 8);

game.computerPlay(d5, d2, d1);

computer = game.getComputer();

assert(computer.whatSpotIsNeededNext() == 9);

game.humanPlay(d6, d2, d3);

human = game.getHuman();

assert(human.whatSpotIsNeededNext() == 10);

game.computerPlay(d1, d2, d3);

computer = game.getComputer();

assert(computer.whatSpotIsNeededNext() == 9);

game.humanPlay(d4, d5, d6);

human = game.getHuman();

assert(human.whatSpotIsNeededNext() == 12);

game.computerPlay(d3, d2, d1);

computer = game.getComputer();

assert(computer.whatSpotIsNeededNext() == 9);

assert(game.isGameOver() == false);

assert(game.determineGameOutcome() == Centennial::GAMENOTOVER); // same logic, testing through the game as it progressed, nextNeeded should be determined by memorizing the previous rolled values.

game.humanPlay(d2, d4, d6);

assert(game.isGameOver() == true);

assert(game.determineGameOutcome() == Centennial::HUMANWONGAME); // forced case by making human winning.